The Best of XCELL

THE NEWSLETTER FOR XILINX PROGRAMMABLE GATE ARRAY USERS

Issues 1 through 6 1989 – 1990

XCELL was started in late 1988 as a quarterly technical newsletter for users of Xilinx Programmable Gate Arrays.

Because our customer base is growing very fast, we found that many new users had not received the first issues of XCELL.

"The Best of XCELL" reprints those articles from previous issues that are still relevant.

Every registered user of a Xilinx development system is automatically a subscriber to XCELL. If you are not a registered user, but would like to be put on our XCELL mailing list, please drop us a note.

Peter Alfke, editor

Table of Contents

Software

Hardware

Using ADI 3.0 Effectively

The May 1990 release of the automatic design implementation software is a significant enhancement over previous versions. It achieves dramatic improvements in terms of both routability and system performance.

We have, however, also received calls from some users saying that they fail to see any improvement, or that the old software gave better results.These problems arise when users over-constrain their design or otherwise limit the software.

To get the best results from the new software, follow these three rules:

• Don't use CLBMAPs unnecessarily.

With older versions of the software it was sometimes necessary to manually map logic such as registers or counters to ensure that a certain order prevailed, e.g. that bits 0 and 1 were mapped together. With bus lines labeled properly, XNFMAP V3.0 will do this automatically. In addition, this version uses better algorithms to combine logic, and from what we've seen, it does as good a job as you would do manually.

• Don't use constraint flags.

Net constraints are the biggest reason why designs do not route completely. To make sure that critical timing paths are met, some designers flag every net in the path as critical. Because of this, APR distorts the placement by putting the critical blocks next to each other, instead of placing them where they should be relative to the rest of the design.

Once the blocks are poorly placed, routing is more difficult. And since the blocks are packed so tightly, longer routing paths are required to go around congested areas to get to the destination pins. When too many flags are put in a constraint file, designs end up running slower than they would have if no flags were used at all!

• Don't place CLBs.

It's best to let APR choose initial block locations. After the design has been placed, you can use XACT to optimize the placement.

What has been a long list of DOs and DON'Ts in previous articles has been shortened to just these three DON'Ts.

The tedious steps of mapping logic, floorplanning, and weighting nets, which were helpful or even necessary for our old software, are now an obstacle for ADI 3.0 to overcome.

So make your job and our software's job easier: skip those steps.

We in Xilinx Applications have been using ADI 3.0 for over a year, starting with early, pre-alpha release versions. We have been able to complete numerous real designs that could not be completed with the previous revisions of our software. Based on this experience, we have developed a methodology for routing tough designs, which is described on the following page. BON

ADI 3.0 Methodology for Tough Designs

1. **Run XNFMAP** with the -M option chosen. This tells the program to ignore CLBMAPs.
2. **Run MAP2LCA** without any command line options.
3. **Delete the .SCP file**. (If IO locations have been specified, keep them but remove everything else.)
4. **Run APR** without any command line options chosen.
5. **Load the design in XACT** and check path delays.
6. If necessary, **weight a few of the nets**, and run APR with the -L option. This will keep the placement locked and re-route the design. This will not take long, since the placement phase is skipped.
7. If the design still isn't routing, there are four ways to solve this problem, depending on the nature of the design:

a) Full Designs

If the CLB utilization of a chip is too high, the design may have trouble routing.

Signs:

- More than 85% of the chip is used.
- Many large net delays.
- Many unrouted pins and nets.

Solutions:

- Run XNFMAP using -CSI options.
- Place blocks and pre-route nets.
- Use a larger part.

b) Over-constrained Designs

Designs with locked IO blocks, many net constraints, or CLBMAPs can be hard to route.

Signs:

- IO assignments in schematics.
- Many net-weight flags in schematics.
- CLBMAP symbols in schematic.

Solutions:

- Run XNFMAP using -M option (ignore CLBMAPs)
- Edit .SCP file to remove constraints and re-run APR.

c) Structured Designs

Full designs with many registers and buses can be hard to route.

Signs:

- Check to see if registers are lined up vertically.
- Check to see if buses are laid out in a logical order.

Solutions:

- Manually line up registers using XACT and re-run APR using the -L option (lock blocks, just route) (Figure 1).

d) Sparse Designs

Designs which use relatively few CLBs (less than 50%) sometimes don't route.

Sign: A partially utilized device won't route.

Solution: In a constraint file, prohibit a few columns in the middle of the chip (Figure 2).

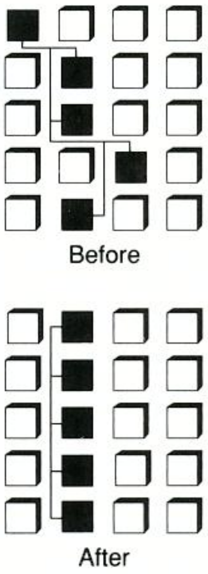


Figure 1

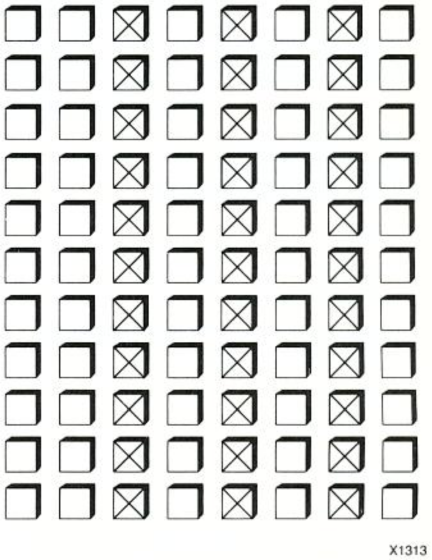


Figure 2

Readback Clarified

The ability to read back configuration data, as well as data stored in flip-flops and latches, is crucial for the exhaustive device testing performed by Xilinx on every device before it leaves the factory.

Most of our customers have no need for this feature, but a few use Readback to verify that the configuration is still proper. This makes sense in applications that require uninterrupted operation, e.g. in telecom where the device may be configured once and then operates for months or years without ever being reconfigured.

To those few engineers who really need the readback feature we apologize for the user-unfriendly interface and the sometimes sketchy documentation.

Here are some important considerations:

Use Readback only when necessary. Less than 1% of all LCA applications use it.

Readback does not interfere with normal LCA operation, but the flip-flop data being read back will be almost impossible to interpret unless the LCA suspends its clocked operation during Readback.

Readback cannot be daisy-chained. Even when the devices were configured in a daisy-chain, they must be read back individually.

Readback data comes out inverted, a configuration 1 becomes a readback 0, and vice versa.

Readback data contains variable flip-flop or latch data in most of the locations that were left unused during configuration. If you want to compare readback data against the configuration file, you must disregard (mask out) these locations as shown below.

Readback has no Preamble, and no second or third stop bit at the end of each frame.

The first frame starts with two dummy zeros instead of the single start bit (1) preceding every other frame. Remember, everything is inverted: Readback start bits are ones, stop bits are zeros.

Before the device is being configured, Readback must be enabled by the MAKEBITS menu:

0 means **never**,

1 means **once**, and

Cmd means **on command**.

Readback is initiated by a rising edge on M0. Rising edges on the CCLK input then clock out the Readback data, using the M1 pin as an output. The first rising edge of CCLK does nothing. The second and third rising edges clock out the two leading dummy zeros. The fourth and subsequent rising edges of CCLK clock out frame information, interspersed with a single 0 for stop at the end of each frame, followed by a single 1 for the start of the following frame. After the last frame stop bit has been clocked out, the M1 pin goes 3-state and further CCLK pulses are ignored.

Verifying Configuration Bitstream

In order to verify the integrity of the LCA configuration, you must compare the Readback bitstream against the configuration bit stream in all those positions not masked out by a 0 in the Mask bitstream.

Configuration bitstream and Mask bitstream have a common format, both are created from the MAKEBITS menu. Since the Readback bitstream format is different, as described above, you must adjust the formats before verification:

Either: Pad the Readback bitstream with preamble, two additional stop bits, and change the two dummy bits preceding the first frame to a normal start bit,

Or, better: Strip the Configuration and Mask bitstreams of the preamble, delete two of the three stop bits and create the two dummy bits at the beginning of the first frame. Always remember that Configuration and Readback have opposite polarity.

After the three bitstreams have been normalized you can perform the verification:

There is an error when (Readback = Configuration) AND Mask = 1.

PA

Park CCLK High

Remember that the CCLK pin must not be held Low for more than 5 µs. Dynamic circuitry inside the LCA can reach an unknown state when capacitors lose charge during excessive CCLK Low time, especially at high temperature, when leakage current is high. If CCLK is held Low after configuration, a subsequent Readback may not function properly, reading back wrong information. Make sure that CCLK has been parked High for several milliseconds before the beginning of Readback.

TCW

Automatic Placement and Simulated Annealing

The Automatic Placement and Routing (APR) program is used during design implementation to perform the placement and routing of an LCA design. The APR program reads an LCA file, generates a new block placement, routes the nets of the design, and writes the result to another LCA file. This article describes the basics of the placement algorithm used by APR, including a description of the options available to the user for controlling the placement process.

Determining the optimal placement of the CLBs and IOBs of an LCA design is a very demanding problem. The number of possible block placements is astronomical, even for the smallest LCA devices, and the number of routing alternatives for a given placement is equally astronomical. Thus, a brute force exhaustive search of all possible placements is impractical.

Within the APR program, a "simulated annealing" algorithm is used to determine the optimal block placement. "Annealing" refers to the crystallization process in which a metal is melted and then slowly cooled back to freezing in order to form highly-ordered crystals with few defects. "Simulated annealing" is an algorithm for finding good solutions to complicated optimization problems (such as the automatic placement of gate arrays) in a manner analogous to the physical crystallization of a metal.

During each iteration of the simulated annealing placement algorithm, two or more blocks are exchanged at random, and the "routability" of the resulting placement is calculated. The routability is expressed in terms of a "routing score", calculated using a formula that includes the length of the routes, the net weighting, and the number of available routing channels as factors. LCA designs are not actually routed during the placement process. The new placement can then be accepted or rejected before going on to the next iteration; if accepted, the new placement becomes the starting point for the next iteration. Accepting only those placements that result in improved routability might cause the design to be trapped in a local optimum, which might be significantly worse than the real optimum.

Therefore, a number of factors determine if a new placement is accepted or rejected. The simulated annealing placement algorithm has two phases: the annealing phase and the quenching phase. During annealing, a new placement that results in better routability is always accepted. However, if the new placement is worse, there is still a probability that it will be accepted, depending on how much worse it is and the current "temperature". The worse the new placement, the less likely it will be accepted; the higher the temperature, the more likely it will be accepted. A random number seed, taken from the system's time-of-day clock by default, also affects the calculation of this probability. As the algorithm proceeds, the design is slowly "cooled" by lowering the temperature. At high temperatures, blocks move freely; as the temperature is lowered, blocks tend to settle into good locations. Once the design is sufficiently cooled, the quenching phase is entered, wherein only better placements are accepted.

By default, APR automatically calculates and assigns a starting temperature sufficient to "melt" the design, that is, a temperature high enough to allow total scrambling of the initial placement. The algorithm also automatically determines the rate at which the temperature is lowered and when the design has cooled sufficiently to allow quenching.

Information describing the progress of the placement algorithm is displayed on the screen while APR is executing, including the temperature, cooling rate, and routing score for the current iteration. The first several iterations typically show a drastically decreasing temperature and a somewhat stable average placement score. As the design cools, the temperature will decrease more slowly and the average and best scores will start to fall. The standard deviation of the scores is the main determinant of how fast the temperature will decrease. As the design nears final placement (temperatures < 100), the temperature will begin to fall more rapidly again.

Command line options specified during the invocation of the APR program can be used to control the placement process, if so desired, as described below:

Option: `-B temp`
Meaning: Set Beginning Temperature

The default is calculated by the APR program, and will be high enough to destroy the placement of the input LCA file. Setting it higher than this makes no sense. But if the initial placement is fairly good, a low starting temperature will prevent the final placement from being radically different from the initial placement. Hence, the temperature can be set to a low number (~100) when the designer wants to limit movement during annealing in order to preserve the structure of the original placement.

Option: `-K rate`

Meaning: Set Cooling Rate
(betwen 5 and 50)

The cooling rate is the percent of temperature decrease from one iteration of the placement algorithm to the next. By default, APR dynamically computes the cooling rate for each iteration. A high cooling rate (~40) can be used for a "quick-and-dirty" placement, especially for sparse designs.

Option: `-Q`

Meaning: Quench Only

This option skips the annealing phase, thereby only allowing placements that are better than the input design. This option is useful for designs whose initial placement is very good, and will result in a significant reduction in APR execution time.

Option: `-E temp`

Meaning: Set Ending Temperature

The default is 1. Lower ending temperatures should not be used. Raising the ending temperature will cause quenching to occur sooner, decreasing execution time but degrading the final placement results. Raising the ending temperature is another means of obtaining a "quick-and-dirty" placement for a sparse design.

Option: `-R seed`

Meaning: Set the Random Number
Seed (1 to 32767)

By default, the seed is derived from the system's time-of-day clock. Successive runs of APR without the -R option will produce different results. Users are encouraged to execute APR several times (using the APRLOOP program), and then choose the best design from the resulting placements.

BKF

The Tilde De-Mystified

Timing values given by XACT or APR are sometimes preceded by the symbol ~, called Tilde by its Spanish name, but really meaning "approximately". How should the user interpret this symbol?

All non-tilde timing values given by XACT or APR are carefully simulated, modeled, and measured worst-case values, guaranteed over the range of processing tolerances and temperature and supply voltage variations. The user can have confidence that no device will ever exceed these values.

The tilde is a disclaimer. It means that the delay is generated by so many concatenated resistor (or pass-transistor) -capacitor elements, that our design and test engineers have less confidence in the accuracy of the model and the repeatability of the timing value. Xilinx cannot guarantee it as an absolute worst-case value.

The number following the tilde is still a conservative specification; most likely the parameter in question is better than this value. But there is not the same guarantee as there is with non-tilde values.

What is the user to do?

If a "tilde-value" is critical to your design, you have two choices:

1. Change the lay-out or routing such that the long uncertain delay is broken up into two "non-tilde" values, either by passing the net through a BIDI or through an unused CLB, or by dividing the net into two branches.
2. Add 25% to the value and ignore the tilde, making the reasonable assumption that this factor 1.25 compensates for the modeling uncertainty.

XC2064 and XC2018 ACLK delay values, though below 10 ns, are sometimes preceded by a tilde. You can safely ignore the tilde in these cases.

There has been a misleading explanation that the tilde indicates propagation delay differences between the rising and the falling edge of a signal. This is not true. Different from original 2.0μ technology XC2000 parts, all new 1.2μ technology devices, and especially the XC3000 family parts, have their delays finely balanced. Our designers have painstakingly adjusted n-channel and p-channel geometries to achieve driving impedances and threshold voltages that guarantee virtually identical propagation delays for rising and falling transitions.

Maybe we have been overly pessimistic and caused unjustified concern with the tilde. But we prefer to be cautious and make a distinction between worst-case guaranteed values and intelligent, albeit conservative, estimates.

PA

The Secrets of "Tie"

Before generating the configuration bit stream, the user has the option to tie or not tie the design. To "tie" means to create additional interconnects that terminate all floating transistor inputs or metal interconnects to well-defined levels or signals.

In a tied design all inputs and interconnects are always High or Low, or are connected to a signal that switches between defined levels. In a non-tied design the unused inputs and pieces of interconnect (there usually are more unused ones than used ones) are left floating. This poses no first order problem, since these inputs are really not used. In this respect it differs from the well-known problem of floating TTL inputs that are supposed to generate a High level.

The problem with undefined input levels in CMOS logic is that they may drift to the midpoint between Vcc and ground, half turning on both the pull-down and pull-up transistor, making a CMOS gate draw measurable Icc. Also, such undefined inputs may be affected by crosstalk from adjacent lines, thus increasing dynamic power consumption.

An untied design is likely to have increased dc and ac power consumption and increased on-chip noise. That's a good enough reason to spend the extra effort to tie every design.

After creating a completely placed and routed LCA design, you must convert the design into a bitstream to download it to the LCA. This conversion is done using the **Makebits** utility in XACT. When you select the Makebits command inside the Makebits menu, you will see three choices: **Tie, Norestore** and **Verbose**. For prototyping, you don't need any of these options; selecting **Done** will generate a bitstream. The Verbose option is only needed when you want the program to send longer messages to the screen.

Always run the **DRC** program before running Makebits. DRC checks for electrical errors in the design. Fix all fatal errors and verify all warnings.

When you are ready to generate a bitstream for production, you should select the **Tie** and **Norestore** options. **Tie** connects all unused inputs and interconnects in the device to legitimate logic levels in order to reduce power consumption and on-chip noise. Tie begins by configuring unused CLBs to create a ground net and then tie these nets to as many places as possible. Any input or interconnect that cannot be tied to these ground nets will be tied to any other existing net unless the net is flagged critical. The **Norestore** option temporarily saves the timing data for the tied design. *Tying a design is not necessary for prototyping but is necessary for production.*

Tying a design may increase the delay of some nets, but usually only by a nanosecond or two. After tying a design, choose the **Norestore** option to save the timing information and then examine the timing of critical nets using the **QueryNet** command under the **Misc** menu to see if the tie operation affected them adversely. If it has, go back to the XACT **Editlca** program and flag the particular net critical with the FLAGNET command under the **Net** menu. This prevents the tie operation from tying to this net and increasing its delay.

The **QueryNet** command has an option called **Tiechange** which compares net timing before and after tie.

The tied design is better electrically, but the many added, logically redundant tie connections make the design very difficult to analyze and virtually impossible to edit. It is, therefore, advisable to store the untied LCA file for future reference. Use the **Restore** command under the **Misc** menu to return the LCA file information to its original state.

You can also save the tied file by using the **Save** command in the XACT Executive. You will be warned that the timing data has not been restored when you exit Makebits and warned that the interconnect is tied down when you try to save the file.

Makebits Tie Problems

If the Makebits Tie program cannot completely tie down a design, it will abort and display messages about what it could not tie. To get a hard copy of these error messages, use your computer's print screen or output file redirect command. Tie often fails because nets are unnecessarily flagged critical. Never flag the global or alternate clock nets critical since these nets are never affected by tie. Go back into XACT Editlca and remove all critical net flags from the design by flagging the nets uncritical and then run Makebits Tie Norestore again. Check to make sure that Tie did not affect the timing adversely. If it did, go back to XACT Editlca and flag critical only those nets whose timing was affected, and then run Makebits Tie again.

Sometimes flagging nets uncritical is not enough. In very rare cases, Tie cannot find a way to connect to some interconnect segments. This some-times affects interconnects near IOB .Q pins in the 3000 family LCAs because they cannot be used to source tie nets. If a certain interconnect (usually a Programmable Interconnect Point or PIP) cannot be tied, Makebits will list the PIPs's location in a format such as:

```
col.A.local.13:row.I.local.10
```

This location can be found in the XACT Editlca editor by typing:

```
find col.A.local.13:row.I.local.10
```

Once the location is found, use the Editnet command to route any nearby net to the PIP that could not be tied, save the design and run Makebits Tie again.

If after removing critical flags and manually editing untied PIPs, the design still does not tie completely, call technical support for assistance.

TCW

Configuring LCAs in Parallel

In the special case where several LCAs contain identical configuration data, they can be configured simultaneously to reduce program size and configuration time. When the program is stored in an XC1736, just make one LCA the Master, all others the Slave, interconnect all CCLKs, and drive all DINs in parallel from the XC1736.

There are no timing problems, even at 1.5 MHz. Between the 666 ns cycle time and the 400 ns access plus 60 ns set-up time, there are over 200 ns available for additional delay. This accommodates at least 250 pF of additional capacitive loading. The 1 MHz max specification on page 2-48 of the '89 Data Book only applies to READBACK; during configuration CCLK can be up to 1.5 MHz.

IOB Options

Pages 2–3 and 2–4 of the Data Book describe the operation of the XC3000 IOBs and their configuration options. This description is not complete: The activation of the passive pull-up is really coupled with the Three-State Enable, giving the fol-lowing four choices:

- Passive pull-up activated, output buffer permanently three-stated (pin is input only, with pull-up).
- Passive pull-up de-activated, output buffer permanently three-stated (pin is input only, no pull-up).
- Passive pull-up de-activated, output buffer active, i.e. three-state control permanently de-activated (pin is ouput only).
- Passive pull-up de-activated, output buffer controlled by three-state control signal (pin can be I/O).

In other words:

The passive pull-up can only be used on pure inputs, not on I/O pins.

The three-state control logic can be permanently disabled, resulting in a permanently active output.

The other four options (OUT INVERT, THREE-STATE INVERT, OUTPUT SELECT and SLEW RATE) are not interdependent, they operate as described.

Helpful Hints for DASH-LCA Users

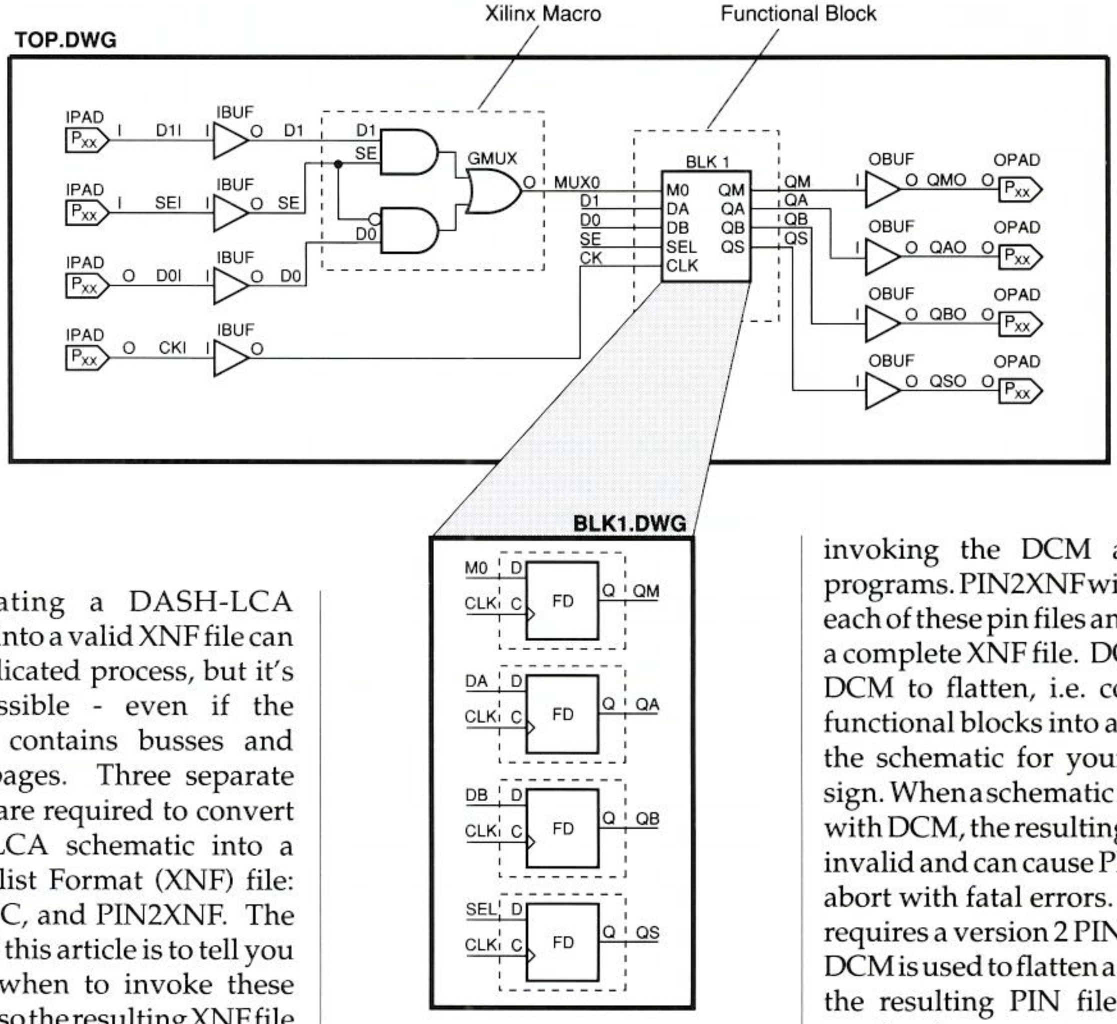


Figure 1

Translating a DASH-LCA schematic into a valid XNF file can be a complicated process, but it's not impossible - even if the schematic contains busses and multiple pages. Three separate programs are required to convert a DASH-LCA schematic into a Xilinx Netlist Format (XNF) file: DCM, PINC, and PIN2XNF. The purpose of this article is to tell you how and when to invoke these programs, so the resulting XNF file can be successfully translated into an LCA file.

Don't be overwhelmed with all the steps in the DASH-LCA design translation process. The Xilinx Design Manager contains an automatic design translation program called XMAKE which invokes all the programs needed to convert your design into an LCA file, including DCM, PINC, and PIN2XNF. All you have to do is push a button and watch XMAKE work!

For more information about DATA I/O's DCM and PINC programs, refer to Volume I of the FutureNet Schematic Designer Manual. Read the "FutureNet Interface" chapter in the XACT Manuals, Volume II to learn more about PIN2XNF.

Processing a Design

A DASH-LCA schematic must be processed with DATA I/O's DCM and PINC programs before it can be converted to a Xilinx Netlist Format (XNF) file. The DCM pre-processor establishes connection data for a schematic drawing and outputs a binary DCM file. PINC, the DASH pin list generator, reads a DCM file and creates an ASCII pin list file (PIN file). Each PIN file contains a listing of the pins on all of the symbols in the schematic and the names of the signals connected to these pins.

You must create a DCM and PINC file for each user-created drawing in your schematic by invoking the DCM and PINC programs. PIN2XNF will then read each of these pin files and generate a complete XNF file. DO NOT use DCM to flatten, i.e. combine all functional blocks into a single file, the schematic for your LCA design. When a schematic is flattened with DCM, the resulting PIN file is invalid and can cause PIN2XNF to abort with fatal errors. PIN2XNF requires a version 2 PIN file; when DCM is used to flatten a schematic, the resulting PIN file format is version 4.

The hierarchical schematic shown in Figure 1 contains one top level drawing (TOP.DWG), one functional block (BLK1), and one XILINX macro (GMUX). The drawing file for BLK1 is appropriately called BLK1.DWG, and is also shown in Figure 1. The following steps are required to create an XNF file for TOP:

1. Create a DCM file for the top level schematic DCM TOP
2. Create a PIN file for TOP.DCM PINC TOP
3. Create a DCM file for the functional block drawing DCM BLK1
4. Create a PIN file for BLK1.DCM PINC BLK1
5. Create an XNF file for the TOP design PIN2XNF TOP

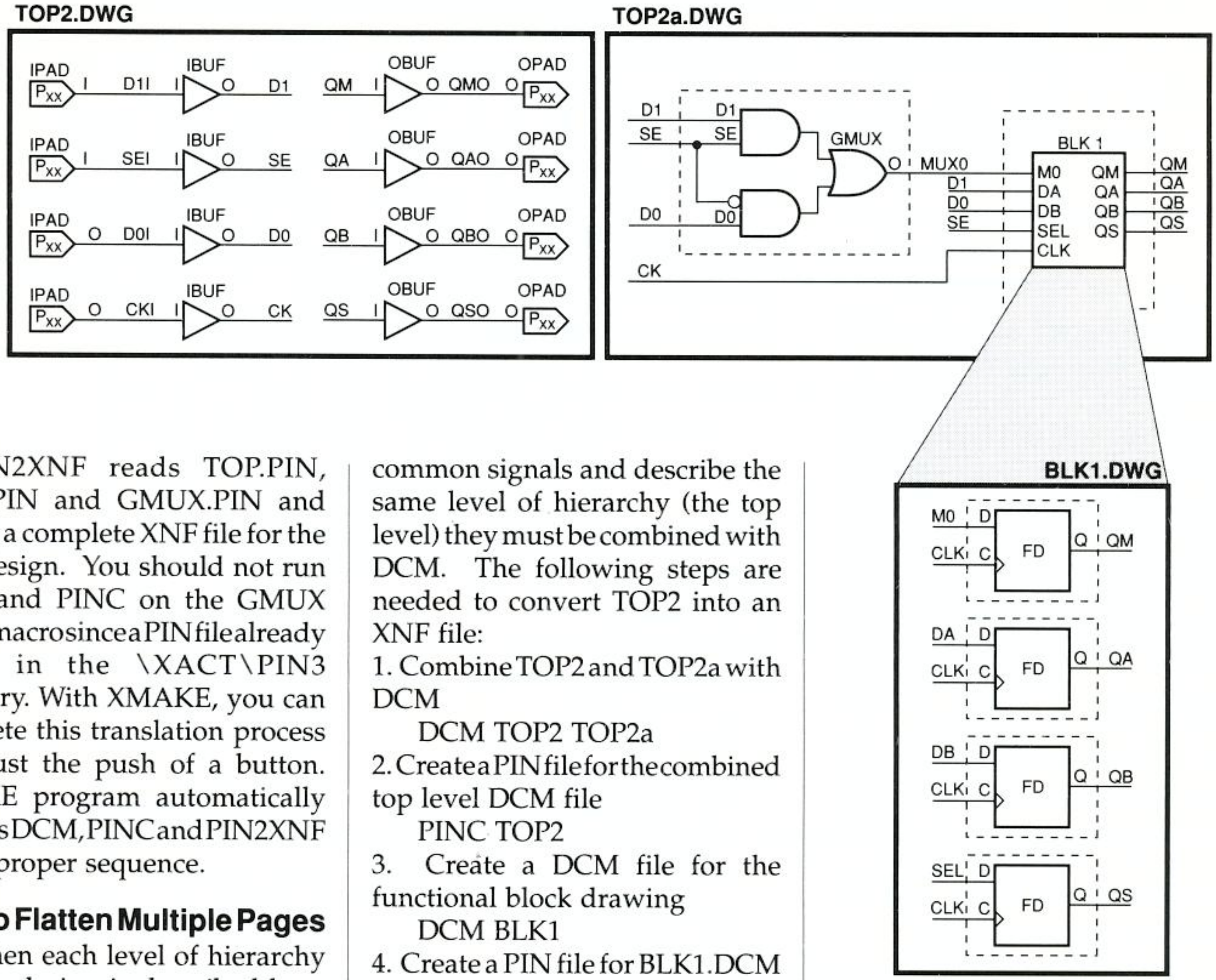


Figure 2

PIN2XNF reads TOP.PIN, BLK1.PIN and GMUX.PIN and creates a complete XNF file for the TOP design. You should not run DCM and PINC on the GMUX Xilinx macro since a PIN file already exists in the \XACT\PIN3 directory. With XMAKE, you can complete this translation process with just the push of a button. XMAKE program automatically invokes DCM, PINC and PIN2XNF in the proper sequence.

How to Flatten Multiple Pages

When each level of hierarchy in your design is described by a single drawing, you must follow the processing procedure outlined above. The TOP design, for example, consists of a single-sheet top level drawing, TOP.DWG, and a functional block which is also described by a single-sheet drawing called BLK1.DWG. When a schematic contains multiple pages at the same hierarchical level, however, you MUST use DCM to combine these pages into a single DCM file. Once multiple pages have been combined, the design translation processes for single-page and multiple-page drawings are identical.

Multiple Pages at the Top Level

The TOP2 schematic shown in Figure 2 has two top level drawings, TOP2.DWG and TOP2a.DWG. Since these drawings contain common signals and describe the same level of hierarchy (the top level) they must be combined with DCM. The following steps are needed to convert TOP2 into an XNF file:

1. Combine TOP2 and TOP2a with DCM

```
DCM TOP2 TOP2a
```

2. Create a PIN file for the combined top level DCM file

```
PINC TOP2
```

3. Create a DCM file for the functional block drawing

```
DCM BLK1
```

4. Create a PIN file for BLK1.DCM

```
PINC BLK1
```

5. Create an XNF file for the TOP2 design

```
PIN2XNF TOP2
```

Multiple Pages for a Functional Block

When a functional block is described by multiple drawings, drawings must be combined with DCM, and the name of this combined file should appear in the functional block symbol with an attribute 8 (file attribute). The functional block MUST contain only one file name (attribute 8). The functional block in TOP3, for example, is described by two drawings, BLK1 and BLK1a. Even though two schematic pages are needed to describe the logic inside the functional block, BLK1 is the only filename which appears inside the symbol. BLK1.DWG and BLK1a.DWG are combined with DCM and the resulting file is used as the input to PINC. This multiple-page drawing, which is shown in Figure 3, is converted to an XNF file with the following commands:

1. Combine TOP3 and TOP3a with DCM

```
DCM TOP3 TOP3a
```

2. Create a PIN file for the combined top level DCM file

```
PINC TOP3
```

3. Combine BLK1 and BLK1a with DCM

```
DCM BLK1 BLK1a
```

4. Create a PIN file for the combined DCM file

```
PINC BLK1
```

5. Create an XNF file for the TOP3 design

```
PIN2XNF TOP3
```

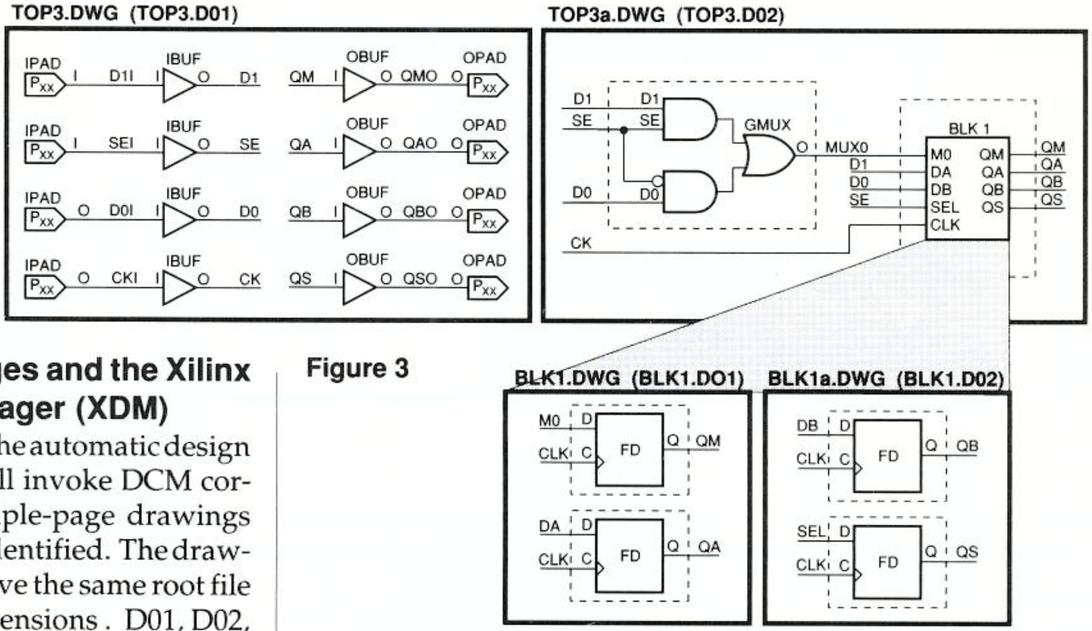


Figure 3

Multiple Pages and the Xilinx Design Manager (XDM)

XMAKE, the automatic design translator, will invoke DCM correctly if multiple-page drawings are properly identified. The drawings MUST have the same root file name with extensions . D01, D02, etc. Consider the TOP3 schematic shown in Figure 3. Before the design is translated, rename TOP3.DWG to TOP3. D01and TOP3a.DWG to TOP3.D02. Also rename BLK1.DWG to BLK1.D01 and BLK1a.DWG to BLK1.D02. XMAKE will recognize that TOP3 and BLK1 contain multiple pages and will invoke DCM with the following command lines:

```
DCM TOP3.D01 TOP3.D02
DCM BLK1.D01 BLK1.D02
```

The resulting DCM files will be called TOP3.DCM and BLK1.DCM; TOP3.DCM will contain logic from both TOP3 schematic pages, and BLK1.DCM will contain logic from both BLK1 schematic pages.

Attributes

Each text string on a DASH-LCA schematic has an associated attribute which provides information needed by post-processors. A signal name, for example, has attribute 5 (signal attribute) and a comment has attribute 0. Many attributes are available with DASH-LCA, but only a small subset can be used on an LCA schematic. Those attributes that are interpreted correctly by PIN2XNF are:

# Definition
0 Comment
2 Location - used to assign symbol names
3 Part - used to define a symbol's type
This attribute appears on Xilinx primitives and is used to define symbol types. **You must not create text with this attribute.**
5 Signal
8 File - attribute assigned to the file name called out inside a functional block.
21 PINO - Input pin
22 PNBT - Bi-Directional pin
23 PINI - Output pin
80 PIN Numbers - Used to define IOB and CLB locations.
81 LCA Part Type - Used to define an LCA part type.
82 XACT Configuration Statements - used on IOB and CLB primitives. 83 Special Options - "FILE=", BLKNM=, FAST I/O, DOUBLE pullups.
84 CLBMAP primitive

Use only these attributes for an LCA design; other attributes might not be interpreted correctly by PIN2XNF. Refer to the "FutureNet Interface" chapter in Volume II of the XACT manuals for more information about the Xilinx-supported DASH-LCA attributes.

Everything You Ever Wanted to Know About Using Busses in DASH-LCA

In DASH-LCA, busses are drawn with "type 2" (/2) lines, and bus names have attribute 5. Individual bus signal names also have attribute 5 and can be connected directly to the bus; no connection dots are needed.

When a bus connects directly to a functional block, a special bus pin is needed. This pin is drawn with the ".=" command in DASH-LCA and must be assigned a pin name, just like any other functional block pin. The pin can have any name (it does not have to match the name of the attached bus) and should have attribute 23 (PINI), 21 (PINO), or 22 (PNBT). An example schematic which contains a bus connected to a functional block is shown in Figure 4.

The functional block in Figure 4 represents the logic in a drawing called BLK.DWG. Since the functional block contains a bus pin called MYBUS, the BLK schematic must contain a bus called MYBUS. **The bus signals which make up MYBUS must have the same names as the signals which make up TOPBUS on the top level schematic.**

In the BLK schematic drawing, MYBUS contains 3 signals, 0, 1, and 2 which match the signal names on TOPBUS. When the TOP4 schematic is translated, MYBUS and TOPBUS are connected and the resulting bus is called TOPBUS.

When DCM is run on the TOP4 schematic, the following error message appears:

```
Warning Bus pin 'MYBUS'
terminates at symbol 11 in
file 'TOP4.DWG'
```

Ignore this message; PIN2XNF will properly translate the drawing.

New Bus Naming Convention

You must follow a new bus naming convention to ensure that PIN2XNF v2.31 or later correctly processes bus signals which connect directly to functional blocks. A bus name (attribute 5) now MUST include the size of the bus and the individual bus signal names MUST be consecutive numbers. The bus in TOP4.DWG, for example, is named TOPBUS<0:2> and the signals are called 0, 1, and 2. The numbering sequence does not have to start with 0, but the smallest number must be listed first in the bus name. (The bus in TOP4.DWG could be named TOPBUS<1:3> with signals 1, 2, and 3; TOPBUS<3:1>, however, is invalid).

Limitation of PIN2XNF V. 2.21

If you are using PIN2XNF v.2.21 (or earlier), you cannot connect a bus directly to a functional block which contains a "FILE=" parameter. Bus signals on the top level drawing are NOT connected properly to bus signals on the lower level by PIN2XNF 2.21. This problem has been fixed in PIN2XNF 2.31.

Do Not Use PIN and POUT Macros

We recommend that you DO NOT use the PIN and POUT macros provided with the DASH-LCA library. When these macros are used, the signal between the pad and the buffer is not named. Since this signal determines the name of the IOB in the LCA design file, its name should be meaning-ful. You can name the net which connects the pad and buffer if you separately invoke the pad and buffer symbols.

CAL

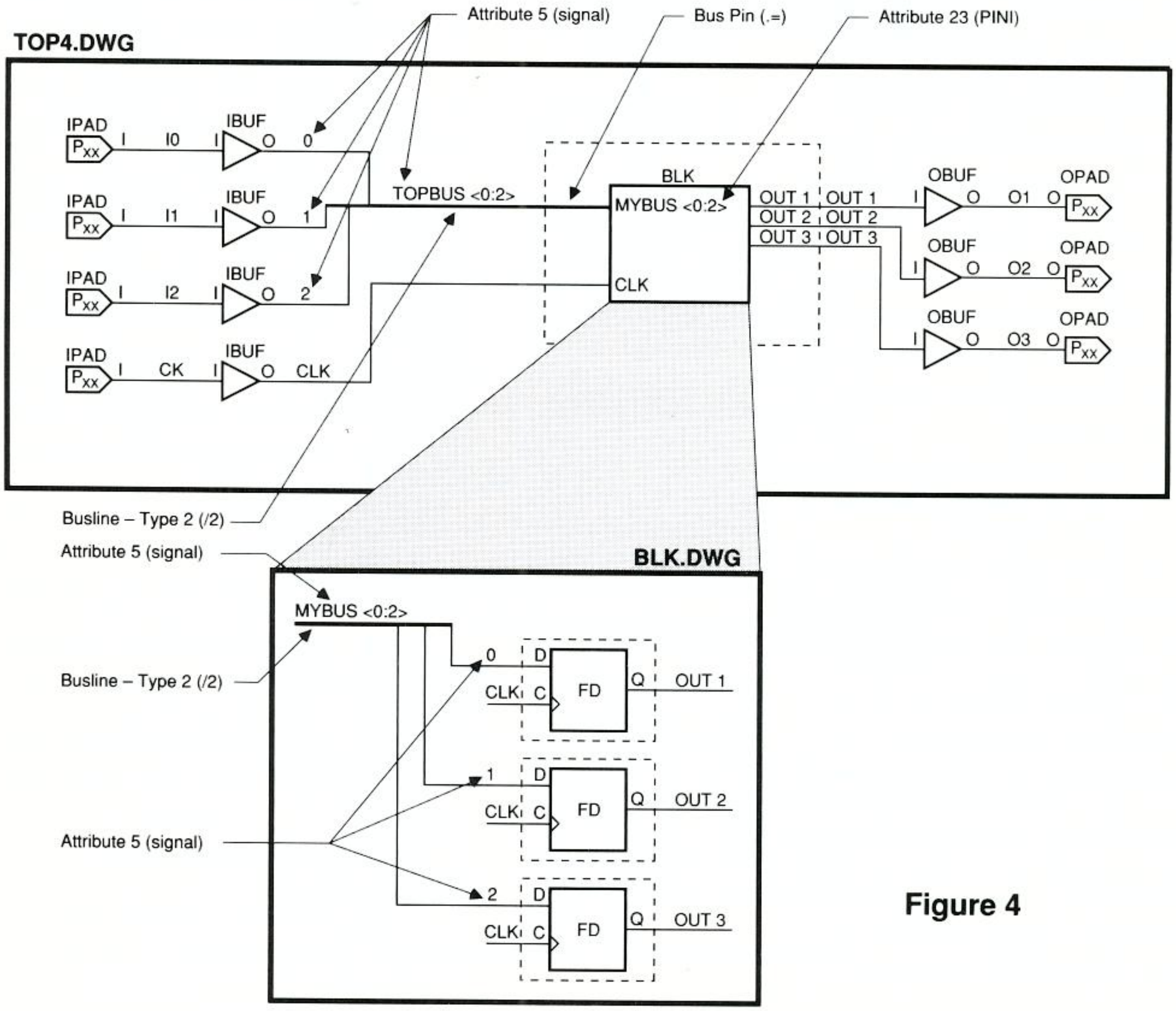


Figure 4

DASH Users: Don't Use DCM to Merge Drawings

Complex LCA designs should be entered in a hierarchical manner, where "modules" on high-level drawings are fully described in lower-level drawings. When entering designs with the DASH schematic editor, the DCM program should not be used to merge the hierarchical drawings.

The translation of a DASH schematic drawing into its corresponding Xilinx netlist file involves three steps. First, the DCM program creates a "drawing connectivity model" from the drawing file. Next, the PINC utility translates the .DCM file into a pinlist file. Lastly, PIN2XNF is used to translate the FutureNet pinlist into the XNF (Xilinx Netlist File) format. (Note: The DCM and PINC utilities are supplied by Data I/O Inc. as part of the FutureNet DASH schematic editor, and are used to generate a FutureNet-compatible netlist from the drawing files. The PIN2XNF program is supplied by Xilinx to translate FutureNet netlists to the XNF format.)

The DCM utility can be invoked in two different manners.

1. DCM can be run on a single-page drawing file by entering

   ```
   DCM filename
   ```

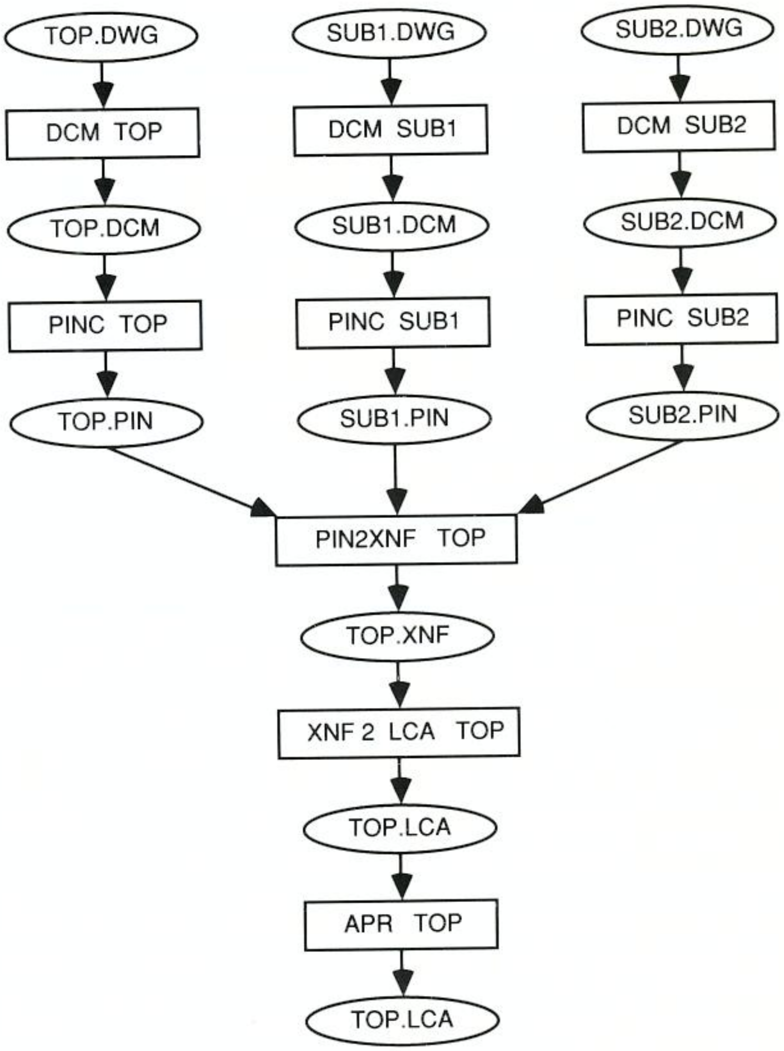
   from the DOS prompt. The .DCM file for that single drawing is then created.

2. Alternatively, if DCM is invoked without a filename, it will prompt the user for the name of the top-level "root" files and the names of all the lower-level drawing files. Using this method, multiple drawings of a hierarchically-structured design can be processed into a single .DCM file by the DCM utility. However, due to structures, using DCM to resolve hierarchical references in this manner may result in unpredictable and undocumented errors from the PIN2XNF program, such as "Undefined pin type in file filename.PIN", and the creation of incorrect XNF files. Xilinx has no control over this, since DCM is a FutureNet program and not a Xilinx program. *This method is not recommended.*

However, this second method of invoking DCM can be used to process a multi-page DASH drawing. For example, if the top-level drawing spans several pages, the filename for each page is entered in response to the prompt for "root" files. However, don't cross hierarchical boundaries using DCM.

The correct methodology for resolving hierarchy for a design entered using DASH is to run DCM and PINC separately for each drawing file, and then let PIN2XNF resolve the hierarchical references. For example, the figure below illustrates the design flow for a design consisting of a top-level drawing named TOP.DWG that references two lower-level drawings, SUB1.DWG and SUB2.DWG.

BKF



Recover from System Crash or Accidental Erasure

One of the most horrifying moments in your life as an LCA designer comes when a power glitch interrupts XACT after you completed hours of work which you had not saved. Even more agonizing is the accidental erasure of an LCA design. These accidents need not ruin your day since the designers of the XACT software have provided ways for you to recover your unsaved work.

If your PC crashes during an XACT editing session, don't panic. Reboot your system and return to the directory you were working in. In the directory you will find the LCA file you were working on. It will be complete up to the point of your last save. You will also find a file with the same name as your LCA file but with a .DBK extension. This is the "data back-up" file and it contains all of the commands you executed in XACT between the last save and the crash of the program. Use the Execute command in XACT and this file to restore your work. First edit the DBK file and remove all commands that caused errors and all exits to DOS. Then call up XACT and load your design into the editor. Once it's loaded, click on the Misc menu and then select the Execute command. You will be asked for the name of the file to execute. Type the name of the data back up file and the .DBK extension and then press ENTER. XACT will then execute all of the commands in the file. When the file is finished executing, the LCA design file will be back to the state it was just before the crash. Save the design immediately!

If you accidentally erase an LCA design file, you can recover your work in a way similar to what is described above. When you save an LCA design file, XACT also saves the previous version of the design in a file of the same name but with a .ODF (old design file) extension and it creates a log file containing all the commands executed between the previous save and this save. The log file has the same name as the LCA file but with a .LOG extension. Use this file to restore your work. First edit the file and remove all commands that caused errors and all exits to DOS. Then call up XACT and load the LCA file into the editor. Once it is loaded, click on the Misc menu and select the Execute command. You will be asked for the name of the file to execute, type the name of the log file with the .LOG extension and press ENTER. XACT will then execute all of the commands in the file. When the file is finished executing, the LCA file will be back to the state of the file that was erased. Save the design immediately and often thereafter!

It is prudent to save LCA design files often and to back them up on floppy disks, but should the unthinkable happen, XACT provides a good way to get you back to work quickly.

TCW

50 New Macros

Name	Description	# of	Speed inMHz	
		CLBs	-70	-100
X74160U	Loadable BCD Up-Counter	6	31	38
X74160D	Loadable BCD Down-Counter	6	31	35
X74161U	Loadable Hex Up-Counter	5	33	41
X74161D	Loadable Hex Down-Counter	5	33	41
X74165S	Synch. Loadable 8-Bit Shift Register	4	55	71
X74165A	Asynch. Loadable 8-Bit Shift Register	8	55	71
C16UPLD	Loadable 16-Bit Up-Counter	19	17	20
C16DNLD	Loadable 16-Bit Down-Counter	19	17	20
C8UDLD	Loadable 8-Bit Up/Down Counter	9	18	22
C16UDLD	Loadable 16-Bit Up/Down Counter	18	11	15
SAR	Successive Approximation Register	1/bit	25	38
BRM	Binary Rate Multiplier	1.5/bit	34	50
X7474	Flip-Flop with Asynch. Preset & Clear	2	55	71
C3SQUARE	Divide-By-Three with 50% Duty Cycle	2	27	33
C5SQUARE	Divide-By-Five with 50% Duty Cycle	2	27	33
M4-1C	Four-to-One Multiplexer in One CLB	1	8ns	7ns
BRLSHFT4	Four-Bit Barrel Shifter	4	17ns	13ns
CBINRIP	Binary Ripple Counter	2	50	62
CDECRIP	BCD Ripple Counter	2	50	62
C5BIT32	5-Bit Divide By 32 Shift Register Counter (Also Divide by 31, 30,...9)	3	38	55
C3BIT8	3-Bit Divide By 8 Shift Register Counter (Also Divide by 7, 6, 5, and 4)	2	50	55
C3BIT8O7	3-Bit Divide By 8 or 7 Shift Register Counter	2	50	55
PHFRCOMP	Phase/Frequency Comparator	2	≥ 10	

I/O Clocking:
Don't Let the Software Confuse You

In all XC3000 family devices each IOB has a choice of two clock sources. These two clock signals are common for each edge of the chip; the same source can of course also be connected to several edges. The clocking polarity of any of these two lines can be selected by configuration, but only for the whole edge of the chip, not for each individual IOB. The XACT user interface incorrectly implies that the designer has the option of selecting clock polarity for each IOB. Not true.

The user can configure each individual IOB storage element as either flip-flop or latch, but the clocking polarities of the two options are interrelated:

If a clock line acts as a rising edge clock for a flip-flop, connecting it to a latch makes the latch transparent while the clock is LOW, latched while the clock is HIGH.

Similarly, a falling edge flip-flop clock becomes an active High Latch Enable. *This relationship is not obvious*, in fact it is surprising, but it optimizes performance. (The latch is really the master portion of the master-slave flip-flop.)

XACT does not make this relationship clear: If flip-flops and latches are to be driven from one clock line, they must be specified with opposite clock polarities: IK for rising edge triggered flip-flop, IKNOT for Low transparent latch, or vice versa. Ignoring this restriction will lead to a fatal DRC error.

Conclusion

If the user needs only one clocking signal per chip edge, there is complete freedom to supply any IOB storage element (flip-flop or latch) with either clock polarity. If the user needs two different clocking signals on any chip edge, then each of them is restricted to one (any one) clocking polarity. Flip-flop and latch options are interrelated in their polarity choice.

Unused Pins

Xilinx Programmable Gate Arrays come with an abundance of user I/O pins, from 58 on the XC2064 to 144 on the XC3090. Many applications leave a few, or even many, of these pins unused, but even unused pins need some attention.

Modern CMOS devices have extremely low input leakage current, perhaps only a few nanoamps. (The 10 µA guaranteed specification represents a testing limitation, not a real input current.)

Left disconnected, such an input could therefore float to any voltage. Clamp diodes prevent excursions above the supply voltage and below ground, thus protecting the input gate from destructive breakdown voltages. This leaves the problem of inputs floating uncontrolled between Vcc and ground.

An input voltage close to the threshold value (1.2V for TTL level-compatibility, 2.5V for CMOS level-compatibility) will turn the input buffer partially on, thus creating a static current path from Vcc to ground and causing static power dissipation. Such a biased buffer also acts as a fairly high gain amplifier, making the circuit very susceptible to noise, crosstalk, ground-bounce and other undesirable disturbances.

It is, therefore, advisable to force unused inputs to a proper logic level.

XC2064 and XC2018

1. Leave unconfigured; externally connect to a High or Low level, or
2. Configure as active output driven by an internally defined signal.

XC3000 Family

Same as above, or

3. Configure as input with internal passive pull-up.

Putting unused I/O to use

An unused XC3000 series IOB can be used as part of the on-chip logic, e.g. as a shift register. Note that the associated package pin must be left free, and that the speed is not as high as it is with internal flip-flops.

Multiple I/O pins can also be used to perform the "wired AND" function in conjunction with an external pull-up resistor.

Don't Overshoot or Undershoot

Our Data Book (pages 2-37, 2-82, 2-101, 2-105, 2-118) explicitly forbids input voltage excursions more than 0.5 V outside the supply voltages (below ground, above Vcc). Hardly anybody would try to violate this with a static voltage or current, but many designs show PC board reflections that sometimes exceed these rather tight limits. A better explanation of the problem is therefore in order:

All CMOS I/O pins are clamped against Vcc and against ground through diodes formed by the respective output transistors. Pure inputs have equivalent protection diodes. These diodes prevent any excessive voltage on the gate of the associated input transistor. Without such protection the input gate might accidentally get charged to a voltage that can rupture the gate oxide and thus destroy the input transistor. All modern MOS devices have such input protection.

What happens when the input voltage exceeds the specified limits?

Below -0.5 V the ground clamp diode will start conducting, above Vcc + 0.5 V the Vcc clamp diode will start conducting. These diodes are fairly big and will clamp hundreds of milliamps with a voltage drop of less than 2 V. The problem is that this clamp current can stray into an area of the circuit where it might upset the internal logic. There is no hard data to quantify this concern, but our circuit designers feel uncomfortable about undefined currents of long duration in parts of the circuit that were not designed for it.

Very high clamp currents (more than 100 mA at elevated temperature, more than 300 mA at room temperature) lasting for milliseconds can cause the parasitic bipolar input transistors to be triggered like an SCR, then conducting unlimited Icc and thus destroying the device. Xilinx devices are extremely resistant to this latch-up.

Conclusion

Try to limit overshoot and undershoot to 0.5 V, the data sheet limit. If these values are exceeded, the clamp diodes will protect the inputs and limit the voltage swing. Large clamp currents of millisecond duration must be avoided at all costs, e.g. by adding current limiting series resistors.

Never drive inputs with active levels above Vcc, even when the Vcc supply is turned off. Strange things might happen during turn-on.

Function Generator Avoids Glitches

The combinatorial logic in all CLBs is implemented as a function generator in the form of a multiplexer, built out of transfer gates. The logic inputs form select the inputs to this multiplexer, while the configuration bits drive the data inputs to the multiplexer.

The Xilinx circuit designers were very careful to achieve a balanced design with similar (almost equal) propagation delays from the various select inputs to the data output.

The delay from the data inputs to the output is, of course, immaterial, since the data inputs do not change dynamically. They are only affected by configuration.

This balanced design minimizes the duration of possible decoding glitches when more than one select input changes. Note that there can never be a decoding glitch when only one select input changes. *Even a non-overlapping decoder cannot generate a glitch problem,* since the node capacitance will retain the previous logic level until the new transfer gate is activated about a nanosecond later.

When more than one input changes "simultaneously," the user should analyze the logic output for any possible intermediate code. If any such code permutation produces a different result, the user must assume that such a glitch might occur and must make the system design immune to it. The glitch might be only a few ns long, but that is long enough to upset an asynchronous design.

If none of the possible address sequences produces a different result, the user can be sure that there will be no glitch.

The designer of synchronous systems generally doesn't worry about such glitches, since synchronous designs are fundamentally immune to glitches on all signals except clocks or direct set/reset inputs.

Worst-Case Input Set-up Time

Timing parameters in programmable devices are more difficult to specify than in fixed-program devices, because the user can affect some parameters through routing.

Inside the LCA, a synchronous design is easy to analyze, because hold time is not an issue, since clock skew is much shorter than the minimum clock-to-Q delay of any CLB. The only concern is for performance: Is the sum of propagation delay and set-up time less than the clock period?

The set-up time at the LCA **input** is more complex, since the clock delay from the clock pad to the internal clock cannot be ignored.

The data sheet specifies the IOB set-up time with respect to its clock (**not with respect to the clock pad!**). The unavoidable delay from clock pad to internal clock must obviously be **subtracted** from the specified set-up time, to arrive at the system set-up time.

What is the maximum value for the input set-up time, and what is its minimum value? Is there a risk for a hold-time requirement?

Maximum Set-up Time

The longest input pad set-up time, the one that determines system performance, is the specified longest IOB flip-flop set-up time minus the shortest clock delay that is consistent with such a long set-up time.

The question is: How well do such delays track?

Here is one **unrealistic** assumption:
"All delays track perfectly. In a given part, at any given temperature and supply voltage, the ratio of any actual parameter value to its specified worst case value is the same constant. "

If this were true, the max set-up time would simply be the difference of the two specified max values for flip-flop set-up time and clock delay.

Here is another **unrealistic** assumption:
"There is no delay tracking. Any parameter can vary between its max and min value, independent of all other parameters."

If this were true, the max system set-up time would be the difference between the specified max flip-flop set-up time and the minimum clock delay, whatever small value that might be.

Both these assumptions are wrong.

The circuits being evaluated reside on one piece of silicon. They were processed together, and they have a common temperature and supply voltage. All delay parameters will, therefore, track reasonably well. But since all parameters do not necessarily depend on the same physical phenomena (resistance, capacitance, threshold voltage etc.) in the same way, they will not track perfectly.

We make the assumption that tracking in any one device will be better than 70%.

All ratios of actual delay to specified worst-case delay for all parameters on the same device at any instant will be within a two-to-one range.

Set-Up and Hold Times

Set-up time describes the requirement for valid input data **prior to** the clock edge.

Hold time describes the requirement for valid input data **after** the active clock edge.

Any particular flip-flop at a particular temperature and supply voltage clocks in the data that happens to be at its input during an extremely narrow picosecond timing window. (If data changes during this narrow window, the flip-flop goes metastable). The width of this window is constant, but its position varies, depending on processing, temperature and Vcc.

The longest set-up time describes the earliest possible position for this window; the longest hold time describes its latest possible position. If no hold time is specified, the set-up time will always be positive, i.e. the window will always be before the clock edge.

These critical set-up and hold time values are often listed in the min column of the data sheet, conforming to an ill-conceived convention established in early 7400 data sheets. Enlightened people have argued for decades that these are really max limits of device parameters, but it has become senseless to fight over form when (hopefully) everybody agrees on the meaning.

•If one delay is close to the specified max value, then all the others will be between 70% and 100% of their respective max values.

•If the relatively slowest parameter is at 50% of its specified max value, then all the other parameters will be between 35% and 70% of their respective max values, etc. (The user should feel safe with this conservative assumption. In reality, parameters track much better than this.)

The longest set-up time is, therefore, the specified max IOB flip-flop set-up time minus 70% of the specified max clock delay.

Example: XC3020-100 using CMOS compatible TCLK input:

Ts max= 17 ns -0.7 •7 ns = **12.1 ns**

Minimum Set-Up Time and Possible Hold Time Requirement

The shortest possible set-up time is the minimum IOB set-up time minus the longest value for the clock delay that is consistent with such a short set-up time. The minimum value for the flip-flop set-up time is not specified, since it is not readily testable. A very conservative guess puts it as short as 10% of the specified max. value. This can only occur at low temperature and high Vcc.

In line with the previous discussion about tracking, the maximum clock delay might then be as long as 14% of its specified max value.

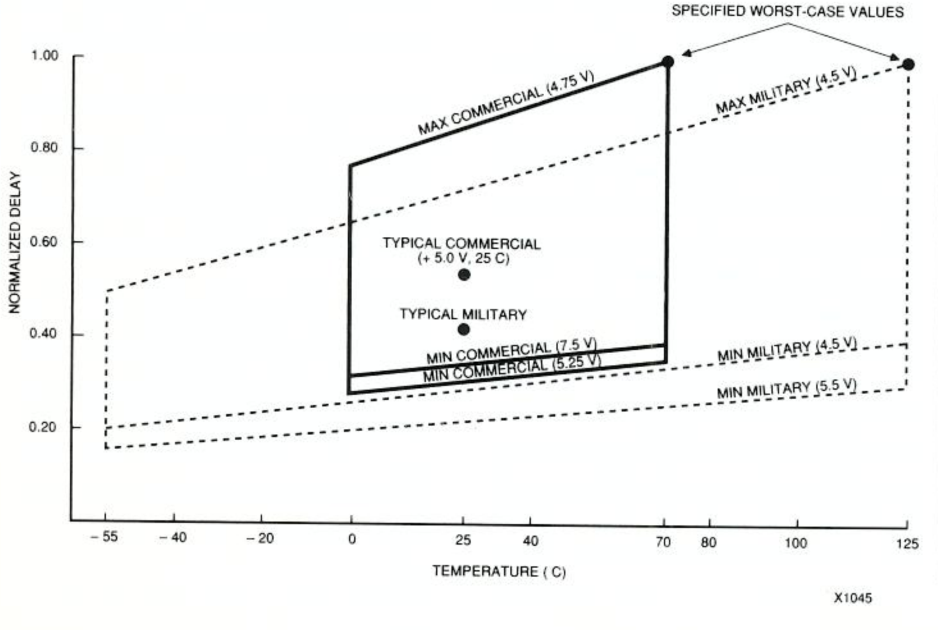
Example: XC3020-100 using CMOS compatible TCLK input:

Ts min= 0.1•17 ns - 0.14•7 ns = **0.7 ns**

This means that the data-to-clock set-up time window on the LCA inputs (pads) is always somewhere between 12.1 ns and 0.7 ns. This is a wide range, but the value is always positive.

There is no hold time requirement. Data may change simultaneously with the clock, provided the clock drives the CMOS-compatible LCA input and uses the Global or Alternate clock distribution network.

PA

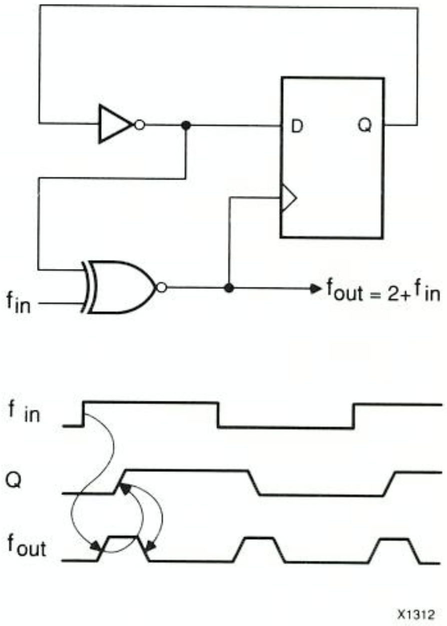


Double the Clock Frequency

A 50% duty cycle input can be doubled in frequency, provided the resulting 2f clock can tolerate a wide variation in duty cycle. The circuit below generates an output pulse in response to each transition on the input.

The output rising edge is delayed one TILO from either input transition. The output High time is the sum of a clock-to-Q delay plus two TILO delays, about 25 ns in a fast part. This output pulse will clock other flip-flops on the same die reliably. (At low temperature and high Vcc the pulse will be shorter, but the flip-flop response is also faster under these conditions)

PA



Rube Goldberg and the Art of LCA Design

Xilinx LCA devices bring gate-array capability to the large number of logic designers who cannot afford the cost, risk, and delay of a masked gate array, but still want to design their own LSI circuits. More new gate-array designs are presently being implemented with Xilinx LCAs than with all other gate-array technologies combined. We love this wide acceptance of our technology, but we are also concerned about the sometimes marginal and even bad logic designs that are being implemented in our parts.

There are exciting arguments that make Xilinx LCAs your favorite choice:

"Build your design in parts, try it out, and modify it if it doesn't work!"
"Fix your mistake before your boss ever sees it!"
"Respond easily to demands created by the market or your competition!"
"Convert an idea you had in the shower to a working chip the same day!"

These are perfectly valid statements, but they do not guarantee design quality. In fact they might actually tempt the designer to be less thorough in the original approach.

"Why perform a careful analysis when I can try it out so easily?"

This attitude can lead to bad design methods and unreliable products. Here are some words of advice, based on over 30 years of systems and logic design experience, and exposure to a few hundred LCA designs over the past two years:

•Always start off with a top-down design. Look at the big picture before you implement the details. Draw a block diagram, step back, and see if you can simplify it by combining functions.

•Trade off complexity against speed. LCA circuitry is quite fast. If your clock runs much slower than 10 MHz, investigate whether you might perform the function time-multiplexed or serially.

•When you design slow logic, don't get careless. The circuitry doesn't know about your low speed; it can still react to nanosecond decoding spikes on the clock or asynchronous reset lines, and might be sensitive to 10ns clock skew problems.

•Be extremely careful with asynchronous inputs. Whenever two asynchronous signals are combined, always (A-L-W-A-Y-S!) perform a thorough worst-case analysis of what happens under the most extreme phase relationships between those inputs.

Use the combination of two asynchronous signals to affect **only one** flip-flop, either to clock it, to reset it, or to synchronize the two signals. One flip-flop will either react or not react to a marginal signal; but several flip-flops might disagree and may cause your system to crash.

Remember, according to Murphy's Law, if something bad **can** happen, it **will** happen, and usually at the most inappropriate moment.

•Never use a decoder to clock or reset a flip-flop or latch asynchronously. Uncontrollable decoding spikes may cause unpredictable behavior that may be temperature-, voltage- or lot-dependent. A classical example: Using the Terminal Count output from a 74161 as a clock is an invitation to disaster.

•Be aware of the metastability problem, but don't be paranoid about it. Read pages 6-20 and -21 of our 1989 Data Book; it shows that an extra 10 ns of tolerable propagation delay can reduce the metastability problem to statistical insignificance.

•Use the global clock distribution network which eliminates all clock skew problems. If you have to distribute a clock signal through general-purpose interconnect and "magic boxes" to several flip flops, always check for clock-skew problems, even if your design is otherwise not time-critical. There usually is an easy cure: Run the clock delays in a direction opposite to the data flow. Clock the most significant part of a counter early, the less significant part later, and all clock skew problems disappear.

•If you have used non-synchronous logic tricks, analyze them very carefully. Check for potential problems with faster parts. Evaluate your design in the fastest LCA that you can buy (presently the -100), to check for potential future problems. There are also two simple ways to change the delay in LCAs mounted on your board: Just vary the temperature or the supply voltage: Heat makes CMOS slow, cold makes it fast; low Vcc makes CMOS slow, high Vcc makes it fast.

If your design fails at high temperature and/or low Vcc, then you are just pushing performance and are running into problems with excessive propagation delays. You might want to improve the routing or use a faster part.

If, however, your design fails at low temperature and/or high Vcc, you have reasons to worry. Take a deep breath and analyze your design for asynchronous abnormalities and for clock skew problems. If you don't fix these problems immediately, they will bite you in the future. CMOS processes are constantly being improved and shrunk. Circuits will get faster, and what was an innocent glitch in a slow part may jeopardize your system in the future.

Logic design is both an art and a science. There are elegant designs and there are kludges; there are rugged designs and there are flimsy contraptions that will inevitably fail sooner or later.

Standard LSI circuits are crafted by experienced logic designers who know what's at stake, are aware of the possible pitfalls and know how to avoid them. And, they simulate their designs and take the time to get it right.

Programmable gate arrays give the user full responsibility for every aspect of the logic design. We hope that the user community is up to this challenge.

The cartoonist Reuben Lucius Goldberg (1883 - 1970) was known for his whimsical drawings of ludicrously intricate machinery meant to perform simple tasks.

PA

The Effect of Marginal Supply Voltage

Since Xilinx LCAs store their configuration in static latches, some users have asked about the integrity of the configuration program under abnormal supply voltage conditions.

Here is a complete description of device behavior during supply ramp-up and ramp-down. Since XC3000 and XC2000 devices behave differently, they are described separately.

XC3000 Family

When Vcc is first applied and is still below about 3 V, the device wakes up in the pre-initialization mode (see fig. 18 on page 2-15 of the Xilinx Data Book). HDC is High; INIT, LDC and D/P are Low, and all other outputs are 3-stated with a weak pull-up resistor.

When Vcc has risen to a value above ~3 V, and a 1 and a 0 have been successfully written into two special cells in the configuration memory, the initialization power-on time delay is started. This delay compensates for differences in Vcc detect threshold and internal CCLK oscillator frequency between different devices in a daisy-chain. The initialization delay counts clock periods of an on-chip oscillator (CCLK) which has a 3:1 frequency range depending on processing, voltage and temperature. Time-out, therefore, takes between 11 and 33 ms for a slave device, four times longer for a master device. This factor of four makes sure that even the fastest master will always take longer than any slave. We assume that the worst case difference between 33 ms and 4x11 ms is enough to compensate for the Vcc rise time spent between the threshold differences (max 2 V) of devices in a daisy chain. Only in cases of very slow Vcc rise time (>25 ms), must the user hold RESET Low until Vcc has reached a proper level.

After the end of the initialization time-out, each device clears its configuration memory in a fraction of a millisecond, then tests for inactive RESET, stores the MODE inputs and starts the configuration process, as described on pages 2-15 through 2-21 of our Data Book. After the device is configured, Vcc may dip to about 3.5 V without any significant consequences beyond an increase in delays (circuit speed is proportional to Vcc), and a reduction in output drive. If Vcc drops into the 3-V range, it triggers a sensor that forces the device back to the pre-initialization mode described above. All flip-flops are reset, HDC goes High; INIT, LDC and D/P go Low, and all other outputs are 3-stated with a weak resistive pull-up. If Vcc dips substantially lower, the active outputs become weaker, but the device stays in this pre-initialization mode. When Vcc rises again, a normal configuration process is initiated, as described above.

The user need not be concerned about power supply dips: The XC3000 family LCA stays configured for small dips, and is "smart enough" to reconfigure itself (if it is a master) or to ask for reconfiguration by pulling INIT and D/P Low (if it is a slave). An XC3000 device will not lock up; the user can initiate re-configuration at any time just by pulling D/P Low or, if D/P is Low, by forcing a High-to-Low transition on RESET .

(continued on next page)

XC2000 Family

date code trailing letter D or E (current production)

When Vcc is first applied and is still below about 3 V, the device wakes up in the pre-initialization mode (see fig. 12 on page 2-67 of the Xilinx Data Book). HDC is High; LDC and D/P are Low, and all other outputs are 3-stated.

When Vcc has risen to a value above ~3 V, and a 0 has been successfully written into a special cell in the configuration memory, the initialization power-on time delay is started in order to compensate for differences in Vcc detect threshold between different devices in a daisy chain. The initialization delay counts clock periods of an on-chip oscillator (CCLK) which has a 3:1 frequency range depending on processing, voltage and temperature. Time-out, therefore, takes between 11 and 33 ms for a slave device, four times longer for a master device. This factor of four makes sure that even the fastest master will always be slower than any slave. We assume that the worst case difference between 33 ms and 4x11 ms is enough to compensate for the Vcc rise time spent between the threshold differences (max 2 V) of devices in a daisy chain. In cases of very slow Vcc rise time (>25ms), the user should wait until Vcc has reached a proper level and then initiate reconfiguration by means of a High-to-Low transition on RESET, as described in more detail below.

After the end of the initialization time-out, each device clears its configuration memory in a fraction of a millisecond, then tests for inactive RESET, stores the MODE inputs and starts the configuration process, as described on pages 2-66 through 2-71 of the 1989 Data Book.

After the device is configured, Vcc is allowed to dip to about 2 V without any significant consequences beyond an increase in delays (circuit speed is proportional to Vcc), and a reduction in output drive.

If Vcc drops below 2 V, but does not dip all the way below 0.1 V, the device might lose its stored information in such a way that it will not recognize a valid Vcc level.

In order to force re-configuration, the user must detect valid Vcc, then force a **High-to-Low transition on RESET while D/P is Low.** RESET must first be High for at least 100 ns, then be forced Low for at least 6 µs, and then be made High again for device configuration. This requirement is described in the left hand bottom paragraph on page 2-72 of the 1989 Data Book.

XC2000 Family

date code trailing letter B or C, or no letter

Early XC2000 devices did not include the circuitry that responds to the High-to-Low transition on RESET, as described above. Once Vcc has dropped below 2 V, these devices may require a very low level, below 0.1 V, on Vcc before they can be re-configured.

PA

Die Organization

Since XACT shows our devices in one specific orientation, and we, therefore, refer to the top, bottom, left and right side of the die, here is a list of the number of horizontal rows and vertical columns for all of our devices.

CLB Organization

	Rows	x	Columns
XC2064	8	x	8
XC2018	10	x	10
XC3020	8	x	8
XC3030	10	x	10
XC3042	12	x	12
XC3064	16	x	14
XC3090	20	x	16

ADI Performance on Different Platforms

We recently ran a large number of real designs on a variety of PCs and workstations. Here is a list of their normalized performance, relative to the '386.
(Performance is the inverse of execution time running ADI 3.0)

PC '386, 20 MHz	1.0
PC '486, 25 MHz	2.8
Sun 3/80	1.5
SPARCstation1+	6.0
DEC DS3100	5.5

The '486 has really moved the venerable PC into the workstation arena, and that's even before we are running ADI in '386 native mode!

Ground Bounce

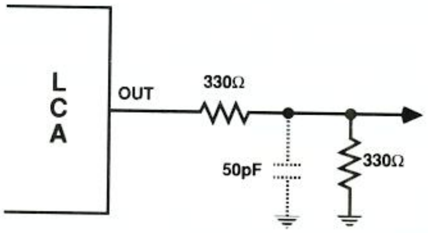
Activating or changing a large number of output pins simultaneously can lead to voltage spikes on the ground and Vcc levels inside the chip.

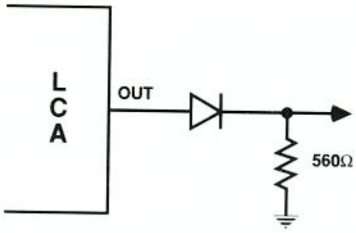
The output current causes a voltage drop in the supply distribution metalization on the chip, in the bonding wires and the lead frame. Worse is the inductive voltage drop caused by the current change over the bonding wire inductance.

This is a well-known problem not only with fast bipolar or CMOS interface devices, but also with high pin-count gate arrays. It is commonly referred to as "ground bounce", because the change in ground potential is more critical than the equivalent change in Vcc potential. (TTL-oriented systems have far less noise immunity at the Low level than at the High level).

Xilinx circuit designers have given the LCAs a very good Vcc and ground distribution metal grid on the chip, as well as double bonding to every supply pin. Packages below 100 pins have two Vcc and ground pin pairs, packages above 100 pins have eight Vcc and ground pin pairs to reduce supply lead resistance and inductance. What can the user do to minimize ground bounce?

- Provide solid Vcc and ground levels. Use multi-layer boards and decoupling. *Wire-wrapping the supply connections is an invitation to disaster.*
- Absolutely always connect all Vcc and ground pins.
- Configure outputs XC3000 slew-limited whenever the required performance allows this. This is the default option. Slew-limited outputs reduce transient amplitude by 75%.
- Use CMOS input levels whenever possible. This increases input noise immunity from less than 1 V to over 2 V.
- Stagger the activation or the change of output drivers by deliberately introduced unequal routing delays.
- Move trouble-causing outputs close to a package ground pin in order to minimize the device-internal voltage drop. Move sensitive inputs, like clocks, close to another package ground pin.
- Finally, if there still is a ground bounce problem on a few outputs, attenuate and/or filter these outputs. A 50% attenuater (330Ω, 330Ω) perhaps combined with a 50 pF decoupling of the center point will reduce V_{OL} and calm it down. Changing the upper resistor to a diode might improve the situation even more.





Three-State vs Output Enable

The control input that causes an IOB output or Long Line driver to go into the high impedance state is called (active High) "Three-State" in Xilinx literature and in XACT. The same signal is commonly known as (active Low) Output Enable or $\overline{\text{OE}}$.

These two signals are identical, i.e. T=$\overline{\text{OE}}$, as explicitly stated on page 2-12 of our Data Book.

To put it more bluntly: T is not an active High Output Enable, rather it is identical with an active Low $\overline{\text{OE}}$.

"Tristate" is a registered trademark of National Semiconductor who pioneered this concept on TTL outputs in the late sixties. The names "Three-state" or "3-State" are ways around this trademark. The name refers to the third state of an output, beyond active High and active Low.

Powerdown Operation

A Low level on the PWRDWN input, while Vcc remains higher than 3 V, stops all internal activity, thus reducing Icc to a very low level:

- All internal pull-ups (on long lines as well as on the I/O pads) are turned off.
- All package outputs are three-stated.
- All package inputs ignore the actual input level and present a 1 (High) to the internal logic.
- All internal flip-flops or latches are permanently reset.
- The internal configuration is retained.
- When PWRDWN is returned High, after Vcc is at its nominal value, the device returns to operation with the same sequence of buffer enable and D/P as at the completion of configuration.

Things to remember

Powerdown retains the configuration, but loses all data stored in the device. Powerdown three-states all outputs and ignores all inputs. No clock signal will be recognized. Any input level between ground and the actual Vcc is allowed.

All internal flip-flops and latches are permanently reset and all inputs are interpreted as High, but the internal combinatorial logic is fully functional.

Things to watch out for

Make sure that the combination of all inputs High and all internal Qs Low in your design will not generate internal oscillations or create permanent bus contention by activating bus drivers with conflicting data onto the same long line.

These two situations are far-fetched, but they are possible and will result in increased power consumption. It is quite easy to simulate these conditions since all inputs are stable and the internal logic is entirely combinatorial, unless latches have been made out of function generators.

Make sure that no applied signal tries to pull any input more positive than the actual supply voltage (Vcc). This would feed Vcc through the input protection clamp diode.

Input Current Is Zero

Some designers keep asking about input current. Let us state bluntly:

The input current is negligible, just nanoamps, if

- the output sharing the same pin is three-stated,
- the internal pull-up option is not activated,
- the device is not in configuration mode where many pins have internal pull-ups as shown on pages 2-32, 2-80, and 2-81 of our '89 Data Book,
- Vcc is above 4V.

If you ever observe our inputs hogging the drive voltage, you must have done something wrong. Make sure you counted the pin number right—and in the right direction, that you configured the device properly, and that Vcc is up. Then use an oscilloscope and multimeter, but please don't use the phone. Our inputs don't draw any current worth talking about.

Don't Pre-Assign Package Pins

In theory, LCAs offer the system designer the option to pre-assign the package pins and lay out the PC board before completing the detailed design of the LCA.

This method works well when the LCA is sparsely populated and, therefore, has the additional routing resources to accommodate an imposed pin-out.

For typical designs this method does not work well. We have seen many cases where the LCA could not be routed with the imposed pin-out, but could be routed once the pin-out was left free. This leads to daughter-board unscramblers or to a re-layout of the PC board, headaches and expenses that the user would like to avoid.

So, as a rule, wait with the pin-out assignment until the LCA has been routed. The exception to this rule are very sparsely populated designs or designs with very limited I/O.

CCLK Max Low Time

Most of the circuitry in our LCAs is static, i.e. the chip will work down to zero clock frequency.

CCLK is the exception. Its circuitry is half-static, half-dynamic and does not tolerate a Low time in excess of 5 microseconds. For very low speed operation, you can stretch the CCLK High time to any desired value, but keep the Low time short.

Internal Bus Contention

The XC3000 family has internal 3-state bus drivers (TBUFs). As in any other bus design, such bus drivers must be enabled carefully in order to avoid, or at least minimize, bus contention. (Bus contention means that one driver tries to drive the bus High while a second driver tries to drive it Low).

Since the potential overlap of the enable signals is lay-out dependent, bus contention is the responsibility of the LCA user. We can only supply the following information:

While two internal buffers drive conflicting data, they create a current path of typically 6 mA. This current is tolerable, but should not last indefinitely, since it exceeds our (conservative) current density rules. A continuous contention could, after thousands of hours, lead to metal migration problems.

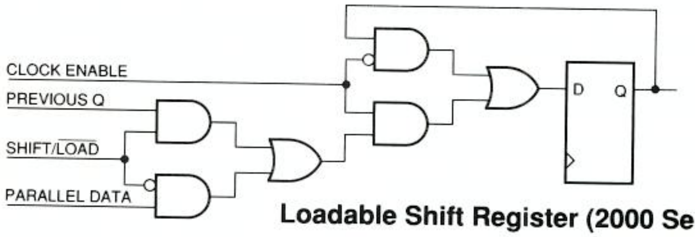
In a typical system, 10 ns of internal bus contention at 5MHz would just result in a slight increase in Icc: 16 bits x 6 mA x 10 ns x 5 MHz x 50% probability = 2.5 mA.

There is a special use of the 3-state control input: When it is directly driven by the same signal that drives the data input of the buffer (i.e. when D and T are effectively tied together as shown on page 2-12 of the Data Book), the 3-state buffer becomes an "open-collector" driver. Multiple drivers of this type can be used to implement the "wired-AND" function, using resistive pull-up.

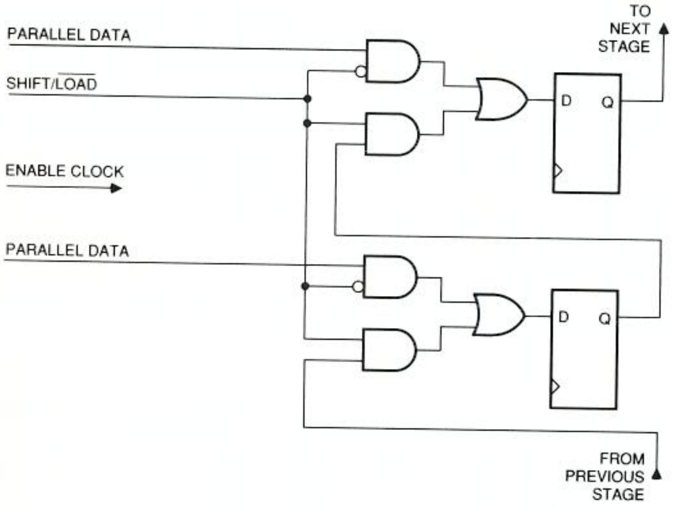
In this situation there cannot be any contention, since the 3-state control input is designed to be slow in activating and fast in deactivating the driver.

The timing calculator actually shows a delay to the D input 2 ns shorter than the delay to the T input. This is no real problem, just a case of misleading modeling. The delay to D does not include the buffer delay whereas the delay to T does.

Connecting D to ground is an obvious alternative, but may be more difficult to route.



Loadable Shift Register (2000 Series)



Loadable Shift Register with Clock Enable

The 2000 Series CLB primitive shown below is a building block for a shift register with synchronous load and clock enable, or for a bidirectional shift register with clock enable but without parallel load.

The 3000 Series CLB primitive shown below is a 2-bit building block for a shift register with synchronous load and clock enable, or for a bidirectional shift register with clock enable but without parallel load.

PA

Just Say NO to Asynchronous Design

Synchronous designs are safer than asynchronous designs, more predictable, easier to simulate and to debug. Asynchronous design methods may ruin your project, your career and your health, but some designers still insist on creating that seemingly simple, fast little asynchronous circuit.

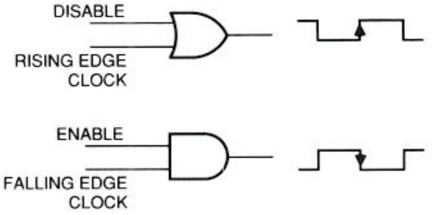
Twenty years ago, TTL-MSI circuits made synchronous design attractive and affordable; fifteen years ago, synchronous microprocessors took over many hardware designs; more recently, synchronous State Machines have become very popular, but some designers still feel the itch to play asynchronous tricks.

The recent popularity of ASICs has created a new flurry of asynchronous designs in a specially treacherous environment: Gate Arrays and Programmable Gate Arrays are being customized at the gate level, and may tempt the designer to develop bad asynchronous habits, especially dangerous since it is very difficult to inspect internal nodes, and impossible to calm them down with capacitive loads, the BandAid of simpler technologies.

Here is a short description of the ugly pitfalls in asynchronous design, documented for the benefit of the inexperienced designer. Veterans are familiar with the problems and may even know their way around them to design safe asynchronous circuits.

Clock Gating

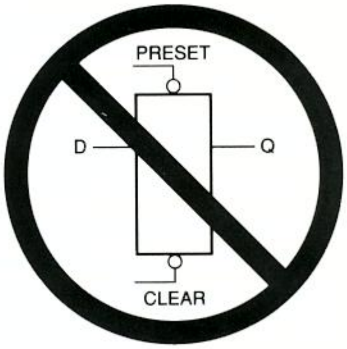
Gating a clock signal with an **asynchronous** enable or multiplex signal is an invitation to disaster. It will occasionally create clock pulses of marginal width, and will sometimes move the clock edge. A **synchronous** signal can be used to gate the clock reliably, as shown below, but this still introduces an additional clock delay, which can cause hold time problems.



Reliable Synchronous Clock Gating

Ripple Counters

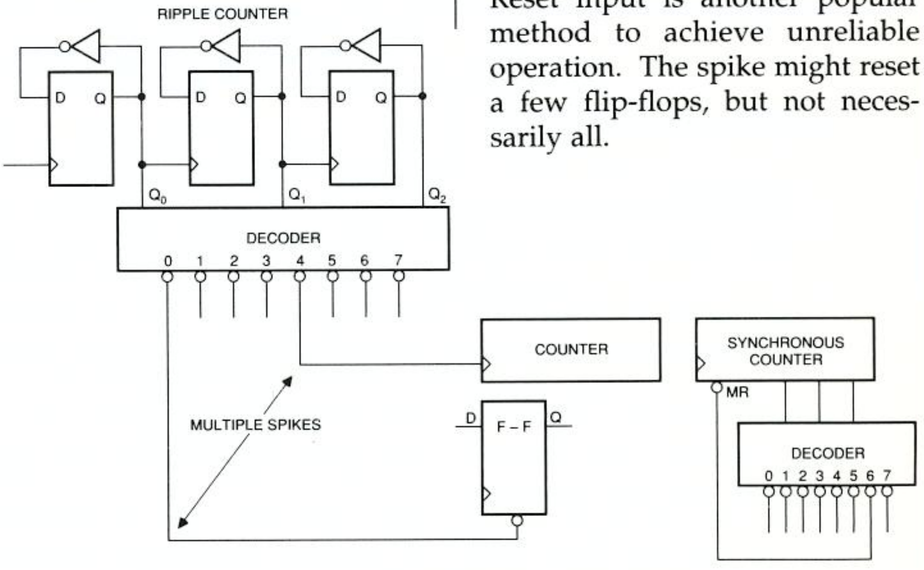
Using the output of one flip-flop to clock its neighbor can generate a binary counter of arbitrary length. The problem occurs when the counter increments from 2^n -1 to 2^n. It takes n delays from the incoming clock to the resulting change in bit n. In a 16-bit counter, this delay will be longer than 100 ns. At a 10 MHz clock rate, certain codes will never exist, the LSB will have changed before the MSB reached its new value. Decoding such a counter will produce dangerous decoding spikes. Note that these spikes are independent of the incoming clock rate. Designers of slow systems are actually most vulnerable to this problem, since they are less sensitive to delicate timing issues.



Decoder Driving Clocks and Reset Inputs

Indiscriminate use of decoder outputs to clock flip-flops or set/reset them asynchronously is one way to invite unpredictable and unreliable operation. The decoded outputs from synchronous counters are even more devious. While the decoding spikes from ripple counters are fairly wide and somewhat predictable, decoding spikes from synchronous counters are entirely the result of small but unpredictable differences in routing and decoding delays.

Using the decoded Terminal Count as asynchronous Master Reset input is another popular method to achieve unreliable operation. The spike might reset a few flip-flops, but not necessarily all.



Unreliable Use of Decoders

Synchronizing One Input in Several Flip-Flops

A single asynchronous input should be synchronized in only one flip-flop. There will be an occasional extra metastable delay as described in the Xilinx 1989 Data Book, pages 6-20, 21. This extra delay is acceptable in all but the very fastest systems. Synchroniz-ing one input in more than one flip-flop is another matter. The set-up times and input routing delays of the various flip-flops will inevitably differ by one or several nanoseconds. Any input change occuring during this time difference will be clocked differently into the individual flip-flops, and the error will last for a full clock period. Synchronize any input with only one single flip-flop!

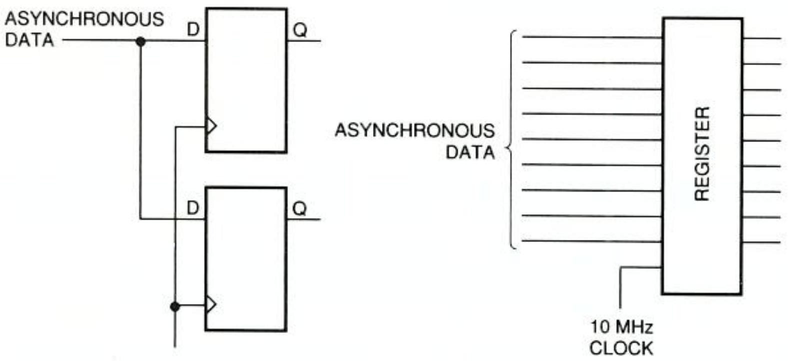
Synchronizing Multiple Inputs in One Register

Synchronizing an asynchronous parallel data word can lead to wrong results when the asynchronous inputs change during the register set-up time. For the duration of one clock period the register might then contain any imaginable mixture of old and new bit values. There is no simple solution, the most popular is to pipeline the result and compare the previous and present values. Any difference declares the data invalid. This operation is sometimes performed in software.

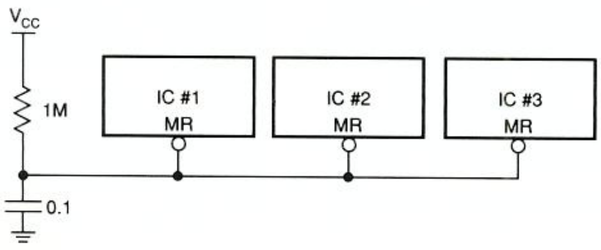
Asynchronous Reset of Multiple Circuits

A simple RC combination, perhaps augmented by a diode, is a popular power-on reset circuit. When it is used to drive several ICs in parallel, the system must accept wide variations in the reset duration.Differences in input threshold voltage will cause some circuits to start operating while others are still being held reset. If that is unacceptable, the RC combination must drive only one IC which, in turn, controls the reset operation of all others.

PA



Dangerous Methods of Synchronizing Asychronous Inputs



Asynchronous Reset of Multiple Circuits

Dot Your T's!

Schematic capture packages have an obsession about details. Some of them insist that a connecting dot be put on every T-joint, even on a connection to a bus. So, even if you think that it's redundant or ugly, put in the dots. It saves you from strange problems later on. One day in the future, we'll have true Artificial Intelligence, and computers will become our servants, not our masters. Until then, dot your T's!

PA

Edit LCAs in the World View

Any command that XACT can execute in the full-size screen can also be performed in the World View. This feature is little known, since most users only show the World View while panning with the cursor.

The alternative is to always show the World View, but it takes up a large portion of the screen, and more often than not, just gets in the way while you're editing. The best solution would be to have the World View appear on command. This is very easy to do. With SHOW NOWORLD set, make the World View appear by pressing a mouse button and moving the cursor. Now stop moving the mouse, press the space bar, and release the mouse button. The World View will stay on the screen. You can now move or swap blocks across the LCA without having to pan around with the mouse trying to find a suitable location.

This feature will be very useful in XACT 3.0, which has new commands that display routing information in the World View.

Design Security

Some Xilinx customers are concerned about design security. How can they prevent their designs from being copied or reverse-engineered?

We must distinguish between two very different situations:

1. The design contains the configuration data in a serial or parallel EPROM or in a microprocessor's memory. This is the normal case.
2. The design does not permanently store a source of configuration data. After the LCA was configured, the EPROM or other source was removed from the system, and configuration is kept alive in the LCA through battery-back-up.

1. In the first case, it is obviously very easy to make an identical copy of the design by copying the configuration data, the devices, and their interconnect patterns. Deleting the part-identifying markings on the top of the ICs would make the copying slightly more difficult, but the main defense is **legal.** The bitstream is easily protected by copyright laws that have proven to be more successfully enforced than the intellectual property rights of circuit designs.

   While it is easy to make an identical copy of the design, (clearly violating the copyright) it is virtually impossible to use the bitstream in order to understand the design or make modifications to it. **Xilinx keeps the interpretation of the bitstream a closely guarded secret.** Reverse-engineering an LCA would require an enormously tedious analysis of each individual configuration bit, which would still only generate an XACT view of the LCA, not a usable schematic.

   The combination of copyright protection and the almost unsurmountable difficulty of creating a design variation for the intended function provides good LCA design security. The recent successes of small companies in reverse-engineering microprocessor support circuits show that a non-programmed device can actually be more vulnerable than an LCA.

2. If the design does not contain the source of configuration data, but relies on battery-back-up of the LCA configuration, then there is no conceivable way of copying this design. Opening up the package and probing thousands of latches in undocumented positions to read out their data without ever disturbing the configuration is impossible.
   This mode of operation offers the **ultimate design security.**

PA

Fixed Modulus Counters / Dividers

Many designs specify a loadable counter when the value to be loaded is actually a constant. In this case the design can be greatly simplified. The most obvious and most general solution uses a variation of the "30-MHz Binary Counter with Synchronous Reset" from page 6-36 of the 1989 Xilinx Data Book. The counter is set to any arbitrary value by changing the RESET to a PRESET on the appropriate bits. The maximum speed calculation has to take in consideration that the time between successive CEP signals may be shorter immediately after a preset of one or more of the least significant bits.

A faster solution uses a variation on the "40-MHz Presettable Counter" described on page 6-38 of the '89 Data Book. Each di-bit uses only two CLBs, as opposed to three in the original version that is loadable with variable data.

PA

Powerdown Pin Must Be High For Configuration

A Low on the $\overline{\text{PWRDWN}}$ pin puts the LCA to sleep with a very low power consumption, typically less than one microwatt. The on-chip oscillator is stopped, and the low-Vcc detector is disabled.

During configuration, the $\overline{\text{PWRDWN}}$ pin must be High, since configuration uses the internal oscillator.

Whenever Vcc goes below 4V, $\overline{\text{PWRDWN}}$ must already be Low in order to prevent automatic reconfiguration at low Vcc. For the same reason, Vcc must first be restored to 4V or more, before $\overline{\text{PWRDWN}}$ can be made High.

PA

Magnitude Comparator: Small, Fast, Expandable

A Magnitude comparator is more complex than an identity comparator, but simpler than an adder or subtracter. A magnitude comparator indicates not only when two operands are equal, but also which one is greater if they are unequal.

PA

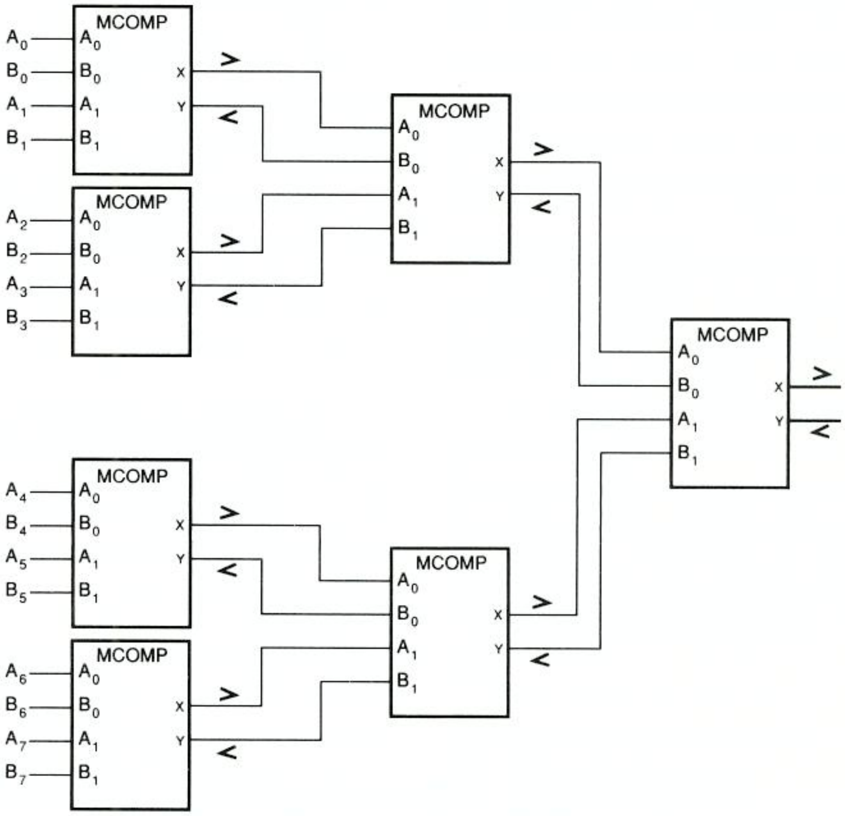
Truth Table

B1	A1	B0	A0	A>B	A<B
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	1	0	0

This truth table is represented by the following equations:

A>B:=A0*~B0*(A1+~B1)+A1*~B1

A<B:=~A0*B0*(~A1+B1)+~A1*B1



Magnitude Comparator Expands to Any Size

LCA Drives Liquid Crystal Display Directly

Non-multiplexed Liquid Crystal Displays (LCDs) must be driven with an ac voltage of 30 to 100 Hz and 10 V peak-to-peak amplitude, **without and dc component**.

Generating this voltage is surprisingly simple in an LCA, using only half a CLB plus one IOB per segment. The back plane of the display is driven by a low frequency (100 Hz) square wave BP, oscillating between 0 and +5 V, and this signal is also used to control the inverting/non-inverting of Data.

When Dout is **in phase** with BP, there is no ac-voltage across the segment, and it looks transparent. When Dout is in **counter-phase** with BP, there is an ac-voltage across the segment, and it appears dark=on.

An additional Light Blanking Input (LBI) can force data to be blank=zero, useful for leading-zero suppression.

NJC

Comparing Data on a Bus

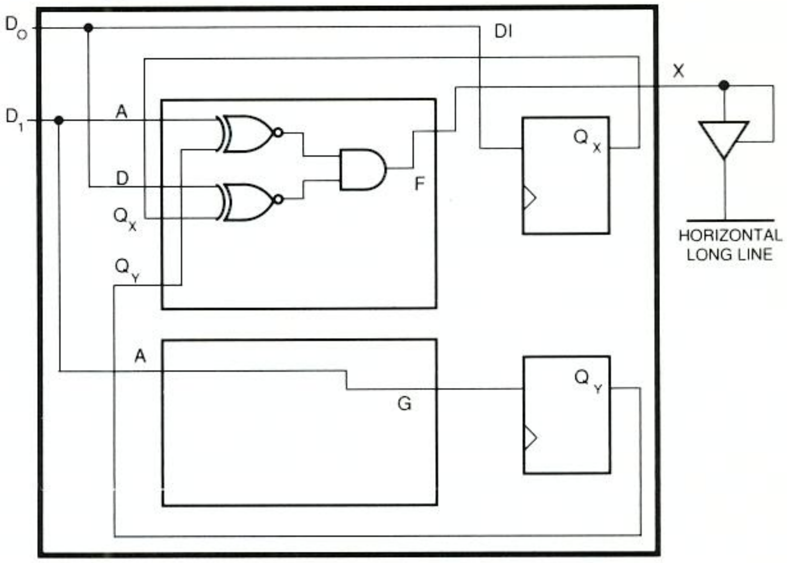
Some systems need to compare variable data on a bus against a value that had previously been loaded from the same bus. Such an identity comparator can store and compare **two data bits per CLB**, then using a Long Line to AND the result.

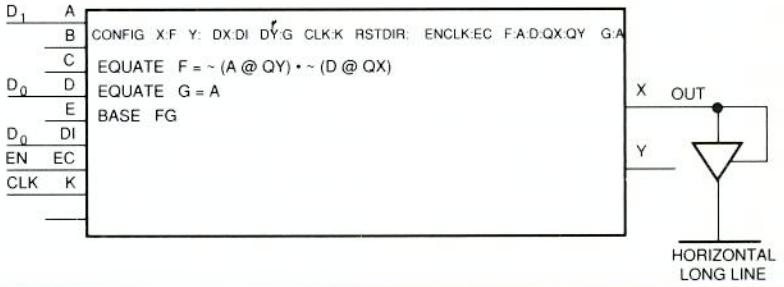
When Enable Clock is active, D0 (through .di) is clocked into Qx, while D1 (through .a) is clocked into Qy. D0 is also routed to the .d input.

The F function generator is brought out and drives the T input of a Long Line buffer. F is High when the two incoming bits match the registered bits.

$F = (\bar{a} \text{ XOR } qy) \bullet (\bar{d} \text{ XOR } qx)$

PA





```
CONFIG  X:F  Y:  DX:DI  DY:G  CLK:K  RSTDIR:  ENCLK:EC  F:A:D:QX:QY  G:A
EQUATE  F = ~ (A @ QY) • ~ (D @ QX)
EQUATE  G = A
BASE  FG
```

Very Fast Accumulator with Pipelined Carry

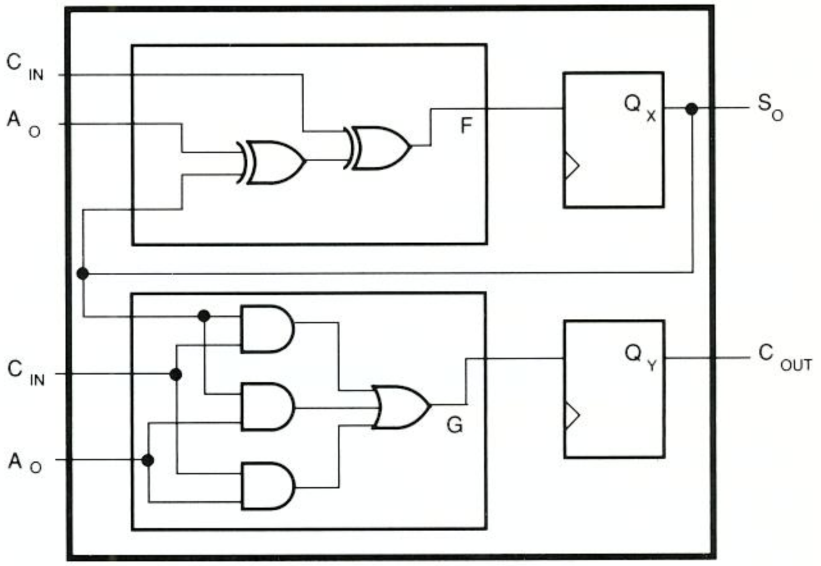
The XC3000 family can implement a very fast (>50MHz) accumulator with pipelined carry.

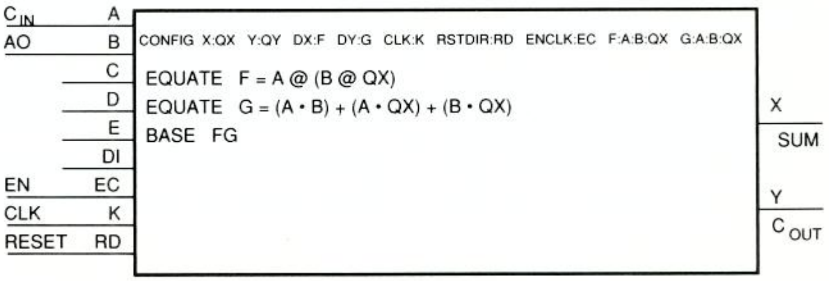
One CLB per bit stores the sum and the carry in its two flip-flops.

Each clock pulse updates the two flip-flops with the result of the addition of incoming operand plus stored sum.

There is, however, one drawback to this pipelined approach: An n-bit accumulator will need up to n-1 additional clock pulses after the last accumulation in order to flush out the carry flip-flops.

PA





```
CONFIG  X:QX  Y:QY  DX:F  DY:G  CLK:K  RSTDIR:RD  ENCLK:EC  F:A:B:QX  G:A:B:QX
EQUATE  F = A @ (B @ QX)
EQUATE  G = (A • B) + (A • QX) + (B • QX)
BASE  FG
```

Programmable Sine Wave Generator

Sine wave frequency synthesizers are used in many applications, like telecom and navigation. A sine wave of programmable frequency can be generated by sequencing through a look-up table in ROM that drives a digital-to-analog converter (DAC).

The simplest and most flexible arrangement uses an accumulator to access the look-up table. (Remember, an accumulator is an adder/register structure that adds an input value to the register content each time it is clocked.) The desired frequency is presented as a constant (K) to the accumulator input. Changing K results in an instantaneous frequency change (as a result of the next clock edge) but no sudden phase change, no "clicks." This is mandatory in modems.

Here is one design example that fits into 30 CLBs, less than half of an XC3020: The objective is to generate any frequency that is an integer multiple of 1 Hz, the highest frequency being around 250kHz. The sine wave look-up table has 64 entries for a 2π (360°) period, i.e. a resolution of 5° to 6°. It represents the amplitude as a 9 bit binary word (8 bits plus sign). These are reasonable parameters, but each of them could easily be modified by an order of magnitude without changing the design concept. The look-up table consists of a 64 x 8 ROM (really a 16 x 8 ROM plus XORs on the address inputs and data outputs) addressed by the 6 most significant outputs of the accumulator.

The ratio of max frequency to frequency resolution determines the size of the accumulator; in this case it is 250kHz ÷ 1Hz = 250K or 18 bits. That would, however, give only one look-up per period at the top frequency; this design, therefore, adds four bits to the accumulator in order to guarantee sixteen look-ups even at 250 kHz. The accumulator; clock rate is then determined by the frequency resolution (1 Hz) and the accumulator length (22 bits): If the accumulator increments by one for every clock period, it must step through the whole look-up table once per second. The clock frequency is, therefore, 2^{22} Hz = 4.194304 MHz.

The four most significant accumulator bits have no data inputs; they can, therefore, be implemented as a counter. The look-up table stores only the first quadrant (90°) of a sine wave, the other three quadrants are generated by reversing the address sequence (XORing the addresses) and/or reversing the sign of the output (XORing the outputs).

Better frequency resolution can be achieved by adding stages to the LSB end of the accumulator (1 CLB for each doubling of the resolution.) Same clock frequency.

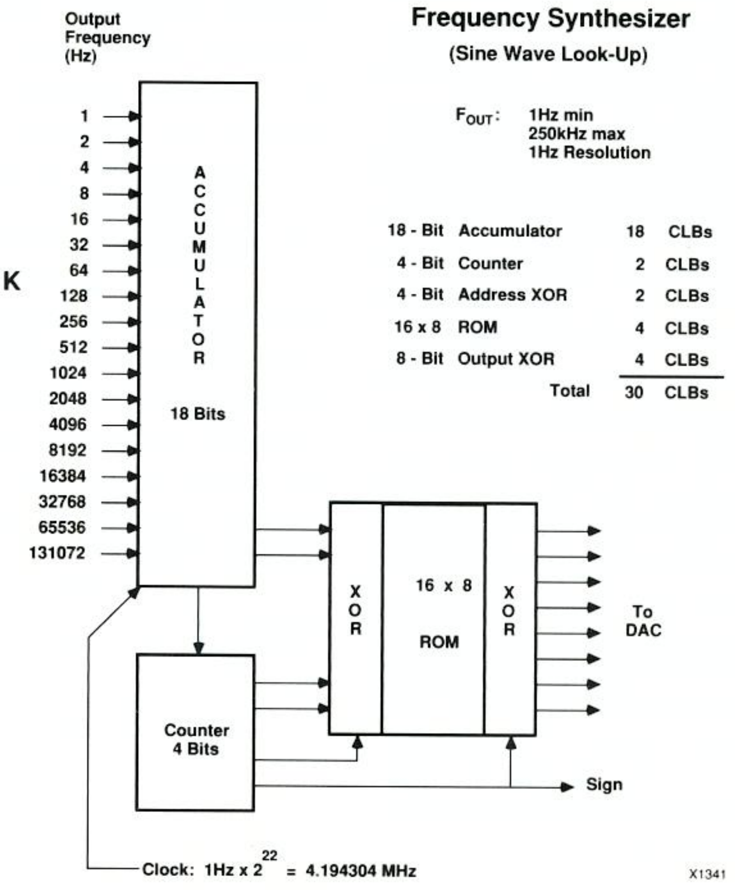
Higher max frequency can be achieved by adding to the MSB end of the accumulator and doubling the clock frequency for every additional bit.

The time-granularity of the look-up table can be doubled to 32 entries per quadrant, increasing the table from 4 CLBs to 8.

The amplitude granularity of the look-up table can be changed in either direction by changing the number of look-up table planes.

Obviously, the look-up table can also store other wave shapes and can be reprogrammed dynamically.

These hints should allow any designer to custom tailor a similar frequency synthesizer. PA



Frequency Synthesizer

(Sine Wave Look-Up)

F_{OUT}: 1Hz min
250kHz max
1Hz Resolution

18 - Bit	Accumulator	18	CLBs
4 - Bit	Counter	2	CLBs
4 - Bit	Address XOR	2	CLBs
16 x 8	ROM	4	CLBs
8 - Bit	Output XOR	4	CLBs
	Total	30	CLBs

No Can Do

Xilinx LCAs offer a wide range of design options and many system-oriented features. There are, however, some restrictions.

Here are the things you should not even try to do in the XC3000 family:

The on-chip input pull-up resistor cannot be used if the pin is configured as I/O, i.e., if the configuration allows the output to be activated. The resistor cannot be used to pull up a 3-stated output,use an external resistor instead.

Bidirectional buses are limited to the length of one Horizontal Long Line. There is no way to interconnect bidirectional buses. There is no pass-transistor between the buses, and two back-to-back amplifiers would latch up.

IOB flip-flops and latches can be reset only by the global RESET package pin that resets every flip-flop and latch on the chip.

Clock polarity is determined at the sources of the IOB's clock line, not at each individual IOB.

IOB latches driven from the same clock line as a flip-flop have a surprising latch enable polarity: Active Low latch enable if the flip-flop clocks on the rising edge, active High latch enable if the flip-flop clocks on the falling edge. This enable polarity must be specified explicitly to avoid a "fatal DRC error".

The two flip-flops in a CLB cannot have separate clocks, clock enable or asynchronous reset inputs.

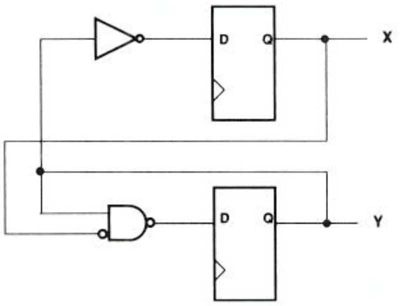
The global clock distribution network cannot be used for anything else but driving CLB and IOB clock inputs. The alternate clock network, however, has limited access to the general purpose interconnects. PA

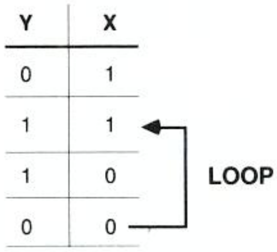
High Speed ÷ 3 Counter in One CLB

Some microprocessors require a 2/3 duty cycle clock, most conveniently and reliably generated by dividing a three times faster crystal oscillator frequency by three.

The design described below uses one XC3000 series CLB to generate a 1/3 High duty cycle signal on the X output, and a 2/3 High duty cycle signal on the Y output. This is just one of many possible implementations. Max clock frequency is 60 MHz in a –70 device.

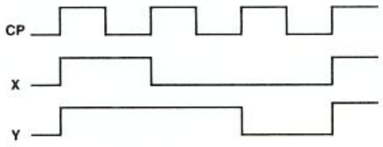
PA





Y	X
0	1
1	1
1	0
0	0

LOOP



Mouse Problem with DASH-LCA?

Some users cannot get the Xilinx-supplied FutureNet Dash-LCA to work properly with their mouse, even though they have specified the mouse properly in their AUTOEXEC.BAT file.

The problem can be as simple as an extra "space" character in your AUTOEXEC.BAT file!

If you edit this file with a word processor, make sure that you do not enter extra spaces in the SET DMOUSE command, not even at the end of the line.

For example, if you are using a Mouse System mouse on the COM1: serial port, your AUTOEXEC.BAT file must include the line:

```
SET DMOUSE=S1
```

There must be only one space on the line, between SET and DMOUSE.

We recommend that you let the INSTALL program modify your AUTOEXEC.BAT file for you, when you first install the DASH-LCA software.

TCW

Chip Select and Write Strobe Timing in Peripheral Mode

There is really only one parameter, the write enable width. It starts when CS0, CS1, and WS are Low and CS2 is High. It ends when this AND condition is no longer met.

This is difficult to convey in a timing diagram, but obvious, once explained. It's the same as in any SRAM:

This Write Enable signal must have a minimum length or width. The Data Book, on page 2-47, specifies 500 ns, but the chip requires, worst case, less than 50 ns. This is another case of an unrealistically conservative timing specification. (Actual characterization data was 9 to 12 ns!)

We are still testing to the old 500 ns limit and cannot change the official spec until we change the test data, but we know that this parameter is similar to the data set-up time, which is specified and tested as 60 ns min.

PA

Max Ten 3090s per Daisy Chain

The infamous 64K byte segment size limitation of the '286 also applies to the Bitstream file. If you try to generate a configuration bitstream of more than 512K bits, the program will bomb ungracefully. The '386 behaves the same way, since we do not (yet) use it in its native mode.

The obvious alternative is to break the daisy chain into several shorter strings, create the bitstreams separately, and combine them in the host.

Note that only Peripheral and Master serial modes can handle configurations longer than 512K bits.

PA

16-Bit Adder Macros

Pages 6-28 and 6-30 of the 1989 Xilinx Data Book describe two 16-bit adder designs without giving detailed CLB maps, placement, and routing information. Both designs have now been implemented in an XC3020 and achieve respectable performance.

The Carry Lookahead adder runs at 13 MHz in a -100 part. The Conditional Sum adder runs at 30 MHz in a -100 part.

The XNF and LCA files are available from our bulletin board in the newly created MACRO section.

BON

Nanowatts, not Microwatts

LCA power consumption in the powerdown state has been somewhat of a mystery. The data book hints at nanowatts (page 2-27), but the published specifications on page 2-39 only guarantee milliwatts.

We tested a representative sample of parts and found the powerdown current at room temperature and 5V mostly below 50 nanoamps. This value is reduced in half at 2.5V, but doubles for every 10°C increase in temperature.

This is good news for battery-back-up. Even the tiniest lithium battery can power an LCA for years.

Why don't we update our guaranteed specification? One reason is the difficulty of measuring very small currents on a high-speed production tester. Another one is the potential yield loss when this parameter happens to be higher. No reason to scrap a part for a parameter that only a few users are interested in.

P.S. There was a small problem in 1988 when some 3000-family devices had a routing- and data-dependent powerdown current of about one milliamp. The problem was traced to a specific diode, and was fixed in early 1989. No part marked after April 1989 shows this larger current.

PA

Print FutureNet Drawings On Your LaserJet

If you have an H-P laser printer (or compatible), you now have some new printing options. A new software package called LASER CONTROL is available from Insight Development Corporation, Emeryville, CA, (415) 652-4115, and might be available at your local PC software store.

For about $90, LASER CONTROL lets you print FutureNet, Schema or OrCAD drawings on any H-P compatible laser printer. It runs approximately ten times faster than the DPLOT software. Also, since LASER CONTROL allows you to shrink an image to half or quarter size, you can print B-size and D-size drawings in an 8.5" x 11" format.

Unfortunately, quarter-size D-size drawings will still be printed in strips on two pages due to limits in the schematic capture program.

To use LASER CONTROL with FutureNet, you must set your printer options in FutureNet accordingly and print the file to disk:

`PRINTOPT W` sets the output size to the widest possible;

`PRINTOPT F` sets the output to the Epson FX format;

`PRINTOPT FILE=<filename>` specifies output filename.

To begin printing the schematic, type `PRINT F`. Your schematic will then be printed to disk. After printing all of your schematics, exit from FutureNet and start the LASER CONTROL software. You may want to verify the correct size for the printout (e.g. full size, half size, or quarter size). The schematic printouts that you "printed" to disk can then be printed on paper by typing:

```
COPY /B <filename> LPT1:
```

LASER CONTROL has a variety of other uses as well. It works with most other PC software.

SKK

XILINX
2100 Logic Drive
San Jose, CA 95124-3450